

# Apple



# Assembly Line

\$1.80

---

Volume 4 -- Issue 12

---

September, 1984

---

### In This Issue...

18-Digit Arithmetic, Part 5. . . . .	2
Faster Amper-routine to Zero Arrays. . . . .	16
Turn an Index into a Mask. . . . .	18
Put Your Messages on the Screen. . . . .	22
Bibliography on Apple Hi-Res Graphics. . . . .	23
Some Great New Books . . . . .	28

### News about Micromation.

Jack Lewis's company, which among other things makes a line of Apple-related products to support the Heathkit Hero robot, has changed its name to Arctec Systems, Inc.

Jack also has a stand alone voice recognition system with an RS-232C interface which may be of interest to some of you. It contains a 65C02 processor, 4K ROM, and 16K of battery-backed-up RAM. Speaker-dependent recognition of up to 256 words or short phrases is possible, with 95-98% accuracy claimed. Arctec's number is (301) 730-1237, in Columbia, Maryland.

### And Some Bad Tidings

The saddest news I have heard lately is of the demise (bankruptcy) of Softalk Publishing. Softalk has been my favorite of all the magazines devoted to the Apple. At this point I do not know how to obtain copies of any of their back issues, or of the books they have published. I assume, and hope, they will be available again soon. With the passing of so many companies, via Chapter 11, many magazines are having great difficulty this year. Unpaid advertising bills then cause a domino effect....

## 18-Digit Arithmetic, Part 5.....Bob Sander-Cederlof

There is a lot of ground to cover in this installment, so I have been forced to use smaller type to squeeze it all in. I want to describe and list the code for the linkage to Applesoft, and for handling arithmetic expressions.

### Loading and Linking to Applesoft

The ampersand (&) statement, according to the Applesoft Reference Manual (page 123, top of page) is

"intended for the computer's internal use only; it is not a proper Applesoft command. This symbol, when executed as an instruction, causes an unconditional jump to location \$3F5. Use reset ctrl-C return to recover."

Not so! The &-statement is intended for adding extensions to the Applesoft language! It does cause a jump by the Applesoft interpreter to \$3F5. If you have not set up any extensions you will get a syntax error when you use "&". But if you have extensions installed, you can work all manner of miracles. DP18 is one such miraculous extension. There are many more around, both in the public domain and in the form of commercial products.

This of course leads to a problem. What if you want to use two or more such extensions? I have written DP18 so that you can chain together one or more additional extension packages as you see fit.

It is very important to decide where the DP18 package will reside in memory. I spent weeks tossing around various options, back when I was designing the DPFF 21-digit package. Of course, at that time, Apples came equipped with anywhere from 16K to 64K RAM; now you can depend on almost all Apples having at least 48K RAM. I still favor the decision I made four years ago, to load the double precision code at \$803, after shifting the Applesoft program far enough up in RAM to leave room.

I have a program I call ML LOADER, which is included on the S-C Macro Assembler disk as a sample program. It performs the function of moving an already-executing Applesoft program up higher in RAM. By including the following line at the beginning of my Applesoft program, I can load DP18 and link it to the & hook at \$3F5:

```
10 IF PEEK(104)=8 THEN PRINT CHR$(4)"BLOADB.ML LOADER"  
:POKE 768,0 : POKE769,30 : CALL770  
:PRINT CHR$(4)"BLOAD DP18"  
:POKE 1014,PEEK(2051) : POKE 1015,PEEK(2052)
```

PEEK(104) looks at the high byte of the starting address of the Applesoft program. Normally Applesoft programs begin at \$801, so PEEK(104)=8. If DP18 has not yet been loaded, then PEEK(104) will still be equal to 8. If it has already been loaded, then the rest of line 10 is skipped.

B.ML LOADER loads at \$300. Its function is to shove the Applesoft program higher in RAM. You POKE the distance to shove into 768 (low byte) and 769 (high byte), than CALL 770. When you wake up an instant later, you have been relocated. The Applesoft program keeps on executing as though nothing happened. Only now there is a gaping hole between \$800 and whatever.

S-C Macro Assembler Version 1.0.....	\$80
S-C Macro Assembler Version 1.1.....	\$92.50
Version 1.1 Update.....	\$12.50
Source Code for Version 1.1 (on two disk sides).....	\$100
Full Screen Editor for S-C Macro (with complete source code).....	\$49
S-C Cross Reference Utility (without source code).....	\$20
S-C Cross Reference Utility (with complete source code).....	\$50
DISASM Dis-Assembler (RAK-Ware).....	\$30
Source Code for DISASM.....	additional \$30
 S-C Word Processor (with complete source code).....	\$50
Double Precision Floating Point for Applesoft (with source code).....	\$50
S-C Documentor (complete commented source code of Applesoft ROMs).....	\$50
Source Code of //e CX & F8 ROMs on disk.....	\$15

(All source code is formatted for S-C Macro Assembler Version 1.1. Other assemblers require some effort to convert file type and edit directives.)

AAL Quarterly Disks.....	each \$15	
Each disk contains all the source code from three issues of "Apple Assembly Line", to save you lots of typing and testing time.		
QD#1: Oct-Dec 1980	QD#2: Jan-Mar 1981	QD#3: Apr-Jun 1981
QD#4: Jul-Sep 1981	QD#5: Oct-Dec 1981	QD#6: Jan-Mar 1982
QD#7: Apr-Jun 1982	QD#8: Jul-Sep 1982	QD#9: Oct-Dec 1982
QD#10: Jan-Mar 1983	QD#11: Apr-Jun 1983	QD#12: Jul-Sep 1983
QD#13: Oct-Dec 1983	QD#14: Jan-Mar 1984	QD#15: Apr-Jun 1984
QD#16: Jul-Sep 1984		

AWIIe Toolkit (Don Lancaster, Synergetics).....	\$39
Quick-Trace (Anthro-Digital).....	(reg. \$50) \$45
Visible Computer: 6502 (Software Masters).....	(reg. \$50) \$45
ES-CAPE: Extended S-C Applesoft Program Editor.....	\$60
Amper-Magic (Anthro-Digital).....	(reg. \$50) \$45
Amper-Magic Volume 2 (Anthro-Digital).....	(reg. \$30) \$25
"Bag of Tricks", Worth & Lechner, with diskette.....	(\$39.95) \$36
Aztec C Compiler System (Manx Software).....	(reg. \$199) \$180

Blank Diskettes (Verbatim).....	2.25 each, or package of 20 for \$40
(Premium quality, single-sided, double density, with hub rings)	
Vinyl disk pages, 6"x8.5", hold two disks each.....	10 for \$6
Diskette Mailing Protectors (hold 1 or 2 disks).....	40 cents each
	or \$25 per 100

These are cardboard folders designed to fit into 6"X9" Envelopes.  
 Envelopes for Diskette Mailers..... 6 cents each  
 QuikLoader EPROM System (SCRG)..... (\$179) \$170

Books, Books, Books.....	compare our discount prices!
"Apple II Circuit Description", Gayler.....	(\$22.95) \$21
"Understanding the Apple II", Sather.....	(\$22.95) \$21
"Enhancing Your Apple II, vol. 1", Lancaster.....	(\$15.95) \$15
Second edition, with //e information.	
"Assembly Cookbook for the Apple II/IIe", Lancaster.....	(\$21.95) \$20
"Incredible Secret Money Machine", Lancaster.....	(\$7.95) \$7
"Beneath Apple DOS", Worth & Lechner.....	(\$19.95) \$18
"Beneath Apple ProDOS", Worth & Lechner.....	(\$19.95) \$18
"What's Where in the Apple", Second Edition.....	(\$19.95) \$19
"6502 Assembly Language Programming", Leventhal.....	(\$18.95) \$18
"6502 Subroutines", Leventhal.....	(\$18.95) \$18
"Real Time Programming -- Neglected Topics", Foster.....	(\$9.95) \$9
"Microcomputer Graphics", Myers.....	(\$12.95) \$12
"Microcomputer Design & Troubleshooting", Zumchak.....	(\$17.95) \$17

We have small quantities of other great books, call for titles & prices.  
 Add \$1.50 per book for US shipping. Foreign orders add postage needed.

\*\*\* S-C SOFTWARE, P. O. BOX 280300, Dallas, TX 75228 \*\*\*  
 \*\*\* (214) 324-2050 \*\*\*  
 \*\*\* We accept Master Card, VISA and American Express \*\*\*

DP18 loads at \$803 and extends well into page \$25. I grabbed 30 pages, moving the Applesoft program to \$2601. It thus clobbers hires screen 1 memory. If you want to use hires screen 2 and the program is too large to fit under it, use POKE 769,88 instead of POKE 769,30 in line 10. This makes the program start at \$6001, and leaves \$2600-3FFF totally unused.

If you want to use other ampersand routines, POKE the link address at locations 2053 and 2054 (\$805 and \$806). If DP18 finds an ampersand command not starting with "DP", it jumps indirectly through this vector. The vector initially contains the address of Applesoft's SYNTAX ERROR routine, but it can be changed to allow using more than one set of &-routines.

### Calling DP18

Whenever you want to execute a DP18 feature, you use the "&DP" statement. If DP18 has been properly connect to the & hook at \$3F5, then the & will send the computer to DP18 (at line 2430 in the listing which follows). At this point DP18 begins to analyze and execute the characters that follow the ampersand.

If the first two characters after the ampersand are not "DP", the program will jump to a vector at \$805 & \$806. This normally points to Applesoft's SYNTAX ERROR routine. However, this location can easily be patched to point to your own ampersand routine.

If the first two characters are correct, DP18 will analyze succeeding statements separated by colons on the same line. There must be a colon immediately after the "&DP" statement. All of the rest of the statements on the line will be executed by DP18, rather than by the normal Applesoft interpreter. If you want to shut off DP18 before the end of the line, two colons in a row with nothing between will do so.

```
150  & DP: INPUT X(0)
160  & DP:Y(0) = X(0) * X(0) * PI: PRINT Y(0) :: GOTO 150
```

It is not necessary that the "&DP:" be the first statement in a line. For example, the following statement will take the square root of a number if the two strings are equal. It uses an Applesoft string comparison, and a double precision square root.

```
170  IF A$ = "SQR" THEN  & DP:Y(0) = SQR (X(0))
```

You can also type double precision statements as direct commands in Applesoft once DP18 has been loaded.

```
]&DP:PRINT X(0): PRINT X(0) ^ 2
```

Four types of statements can be executed by the DP18 package: assignment, INPUT, PRINT, and IF statements. INPUT and PRINT statements will be covered in a later installment.

The DP18 IF statement evaluates a logical expression in 18-digit precision, and then reverts to normal Applesoft processing:

```
180 &DP : IF A(0) < 1.52345678976543 THEN X = 3
```

The DP18 assignment statement takes two forms: real assignment, and string assignment. String assignment is used to convert DP18 values to strings, so

that they can be used by normal Applesoft:

```
190 &DP : A$ = STR$ (X(0))
```

Real assignments are the normal computational statements, like:

```
200 &DP : A(0) = (4*PI*R(0)^3)/3
```

#### DP18 Variables

All variables referenced by DP18 must consist of two adjacent array elements. The array must be a REAL array, that is, it must not be INTEGER or STRING.

Remember that Applesoft array subscripts begin with 0 and go up to the limit defined in the DIM statement. An array dimensioned "3,11,11" has three dimensions. The first runs from 0 to 3; the second from 0 to 11; and the third also from 0 to 11. It could contain  $4 \cdot 12 \cdot 12 = 576$  real elements, or  $2 \cdot 12 \cdot 12 = 288$  double precision elements.

Applesoft arrays are stored in memory with the leftmost subscript varying the fastest. For example, in the array XY(3,10,10), element XY(0,j,k) comes immediately before element XY(1,j,k). Therefore you may, in effect, create an array of double precision values by merely prefixing an extra dimension to the dimension list.

If you wish to set up separate variables, you may do so by dimensioning them to have two real elements each. For example, the statement

```
10 DIM A(1),B(1),C(1),X(1)
```

will set up four separate variables for use with DP18. You reference the variables within double precision statements with the subscript 0. For example:

```
20 & DP:X(0) = (A(0) + B(0)) * C(0)
```

Note that you don't have to dimension these variables, since Applesoft will default to a dimension of 10. However, it is a good idea to dimension all double precision variables because it saves memory (only 2 real elements are allocated instead of 11) and it makes it easier for someone else to follow your program.

If you wish to create an array of double precision values, you do so by dimensioning the array with one extra dimension. The extra dimension comes first and should be "1"; this dimension generates two real items, or one double precision item. For example,

```
10 DIM A(1,12),B(1,5,6)
```

creates two arrays that can be used for double precision values. The array A can be thought of as an array of 13 double precision values from A(0,0) to A(0,12). The array B could store 42 double precision values from B(0,0,0) to B(0,5,6). If you always remember to use one extra dimension, to put that extra dimension first, to set that dimension to "1", and to refer to items with the first subscript = 0, then you will succeed in using DP18.

## DP18 Constants

Double precision constants are entered in the same way as single precision constants. The differences between standard Applesoft and the DP18 constants are that DP18 converts and stores 18 significant digits rather than 9, and that exponents may be in the range of +/- 63 rather than +/- 38.

Conversion of constants is very fast in DP18. DP18 will convert constants over 4 times faster than normal Applesoft, even using more digits! It is quicker to convert a constant than it is to find and use a DP18 variable, especially multi-dimensioned variables. This is completely opposite from normal Applesoft, where variables are quicker than constants.

## Conversion Between Single and Double Precision

You will often need to convert a single precision value into a double precision one for purposes of computation. This is easily done by first converting it to a string and then using DP18's VAL function as shown here.

```
100 REM CONVERT X TO DOUBLE PRECISION VALUE
110 DIM DP(1)
120 INPUT "VALUE TO BE CONVERTED? ";X
130 &DP:DP(0) = VAL ( STR$ ( X ))
140 &DP: PRINT DP(0)
150 GOTO 120
```

You will also want to convert from double precision back to single precision. This also involves converting to a string, but takes more than one statement.

```
100 REM CONVERT DP(0) TO SINGLE PRECISION VALUE
110 DIM DP(1)
120 &DP:INPUT "VALUE TO BE CONVERTED? ";DP(0)
130 &DP:A$ = STR$ ( DP(0))
140 X = VAL ( A$ ) : PRINT X
150 GOTO 120
```

Note that lines 130 and 140 could be combined onto one line if there were two colons separating the statements. See the section on functions for more information about the STR\$ and VAL functions.

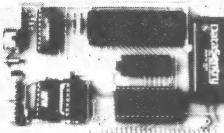
## DP18 Arithmetic Expressions

Expressions in DP18 are very much like expressions in Applesoft. Except for AND and OR, they are evaluated using the standard rules of precedence as found on page 36 of the Applesoft manual. AND and OR have the same precedence in DP18 and are executed left to right. The order of precedence is listed below. Operations on a higher line are executed before operations on a lower line. Operators on the same line are executed left to right.

```
( ) function calls
+ - NOT      unary operators
^
* /
+ -
< > = <= >= => =< > >
AND OR
```

# Apple Peripherals Are All We Make

## That's Why We're So Good At It!



- Absolutely, positively, totally PRO-DOS and DOS 3.3 compatible.
- Time in hours, minutes, seconds and milliseconds (the ONLY PRO-DOS compatible card with millisecond capability).
- 24 hour military format or 12 hour with AM/PM format.

- Date with year, month, day of week and leap year.
- The easiest programming in BASIC..
- Eight software controlled interrupts so you can run two programs at the same time (many examples are included).
- Compatible with ALL of Apple's languages. Many sample programs for machine code, Applesoft, CP/M and Pascal on 2 disks.
- On-board timer lets you time any interval up to 48 days long down to the nearest millisecond.
- Rechargeable nickel-cadmium battery will last over 20 years.
- Two BSR/serial ports for future expansion.

### THE NEW TIMEMASTER II H.O.

Full emulation of all other clocks. Yes, we emulate Brand A, Brand T, Brand P, Brand C, Brand S and Brand M too. It's easy for the H.O. to emulate other clocks, we just drop off features. That's why the H.O. can emulate others, but none of the others emulate us.

The Timemaster II H.O. will automatically emulate the correct clock card for the software you're using. You can also give the H.O. a simple command to tell it which clock to emulate. This is great for writing programs for those poor unfortunate that bought some other clock card.

Of course, most programs will use the Timemaster II H.O. in its native mode, but it's comforting to know that you can use programs written for other products without any modification.

### REMOTE CONTROL

Our BSR X-10 interface option for the H.O. allows you to remotely control up to 16 lights and electrical appliances through your BSR X-10 home control system in your home or office. You're already wired because a BSR system sends its signals over regular 120 volt wiring. That means you can control any electrical device in your home or office without any additional wiring.

**PRICE \$129.00 BSR Option \$49.00**

### MemoryMaster IIe 128K RAM Card

- Expands your Apple IIe to 192K memory.
- Provides an 80 column text display.
- Compatible with all Apple IIe 80 column and extended 80 column card software (same physical size as Apple's 64K card).
- Can be used as a solid state disk drive to make your programs run up to 20 times FASTER (the 64K configuration will act as half a drive).
- Permits your IIe to use the new double high resolution graphics.
- Automatically expands Visicalc to 95K storage in 80 columns! The 64K config. is all that's needed. 128K can take you even higher.
- PRO-DOS will use the MemoryMaster IIe as a high speed disk drive.
- The 64K MemoryMaster IIe will automatically expand Apple Works to 55K storage. The 128K MemoryMaster IIe will expand Apple Works to 101K storage.
- High speed disk emulation for BASIC, Pascal and CP/M is available at a very low cost. NOT copy protected.
- Documentation included, we show you how to use all 192K.

If you already have Apple's 64K card just order the MEMORYMASTER IIe with 64K and use the 64K from your old board to give you a full 128K. (The board is fully socketed so you can simply plug in more chips.)

MemoryMaster IIe with 128K	\$249
Upgradeable MemoryMaster IIe with 64K	\$169
Non-Upgradeable MemoryMaster IIe with 64K	\$149

### Z-80 PLUS

- TOTALLY compatible with ALL CP/M software.
- The only Z-80 card with a special 2K "CP/M detector" chip.
- Fully compatible with Microsoft disks (no pre-boot required).
- Specifically designed for high speed operation in the Apple IIe (runs just as fast in the II+ and Franklin).

- Runs WORD STAR, dBASE II, COBOL-80, FORTRAN-80, PEACHTREE and ALL other CP/M software with no pre-boot.
- A semi-custom I.C. and low parts count allows the Z-80 Plus to fly thru CP/M programs at a very low power level. (We use the Z-80A at fast 4MHz.)
- Does EVERYTHING the other Z-80 boards do, plus Z-80 interrupts. Don't confuse the Z-80 Plus with crude copies of the Microsoft card. The Z-80 Plus employs a much more sophisticated and reliable design. With the Z-80 Plus you can access the largest library of software in existence. Two computers in one and the advantages of both, all at an unbelievably low price.

**PRICE \$139.00**

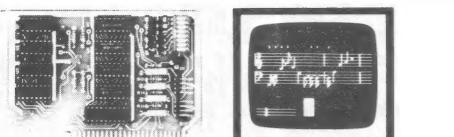
### VIEWMASTER 80

	PRICE	BUILT-IN VIDEX COMPATIBILITY	WHITE KEY VIDEX COMPATIBILITY	LINE INPUTS	LINE OUT	VIDEO IN	VIDEO OUT	RIGHT PNL VIDEX COMPATIBILITY	VIDEO OUTPUT	80 COLUMN COMPATIBILITY	INVERTER OPTION
VIEWMASTER	159	YES	YES	YES	YES	NO	NO	YES	YES	YES	YES
SUPERVIEW	MORE	NO	YES	NO	NO	NO	NO	NO	NO	NO	YES
WIZARD80	MORE	NO	NO	NO	NO	NO	NO	NO	NO	NO	YES
VISION80	MORE	YES	YES	NO	NO	YES	NO	NO	NO	NO	NO
OMNIVISION	MORE	NO	YES	NO	NO	NO	NO	NO	NO	YES	YES
VISIACALC	MORE	YES	YES	NO	NO	NO	NO	NO	NO	NO	YES
VISIACALC80	MORE	YES	YES	NO	NO	NO	NO	NO	NO	NO	YES
VIDEOTERM	MORE	YES	YES	NO	NO	NO	NO	NO	NO	NO	NO
VIDEOTERM80	MORE	NO	YES	YES	NO	YES	YES	NO	NO	NO	YES

The VIEWMASTER 80 works with all 80 column applications including CP/M, Pascal, WordStar, Format II, EasyWriter, Apple Writer II, VisiCalc, and all others. The VIEWMASTER 80 IS THE MOST compatible 80 column card you can buy at ANY price!

**Thousands SOLD at \$179 NOW ONLY \$159.00**

### SUPER MUSIC SYNTHESIZER - END MOCKINGBOREDOM



- Complete 16 voice music synthesizer on one card. Just plug it into your Apple, connect the audio cable (supplied) to your stereo, boot the disk supplied and you are ready to input and play songs.
- It's easy to program music with our compose software. You will start right away at inputting your favorite songs. The Hi-Res screen shows what you have entered in standard sheet music format.

- Now with new improved software for the easiest and the fastest music input system available anywhere.
- We give you lots of software. In addition to Compose and Play programs, 2 disks are filled with over 30 songs ready to play.
- Easy to program in Basic to generate complex sound effects. Now your games can have explosions, phaser zaps, train whistles, death cries. You name it, this card can do it.
- Four white noise generators which are great for sound effects.
- Plays music in true stereo as well as true discrete quadraphonic.
- Full control of attack, volume, decay, sustain and release.
- Our card will play notes from 30Hz to beyond human hearing.
- Automatic shutdown on power-up or if reset is pushed.
- Many many more features.

**PRICE \$159.00**

Our boards are far superior to most of the consumer electronics made today. All I.C.'s are in high quality sockets with mil-spec. components used throughout. P.C. boards are glass-epoxy with gold contacts. Made in America to be the best in the world. All products work in the Apple IIe, II+, II+, and Franklin. The MemoryMaster IIe is IIe only. Applied Engineering also manufactures a full line of data acquisition and control products for the Apple, A/D converters and digital/O cards, etc. Please call for more information. All our products are fully tested with complete documentation and available for immediate delivery. All products are guaranteed with a no-hassle THREE YEAR WARRANTY.

Send Check or Money Order to:

APPLIED ENGINEERING  
P.O. Box 798  
Carrollton, TX 75006

Call (214) 492-2027

8 a.m. to 11 p.m. 7 days a week  
MasterCard, Visa & C.O.D. Welcome  
No extra charge for credit cards

Texas Residents Add 5% Sales Tax  
Add \$10.00 If Outside U.S.A.

These all work the same as they do in Applesoft, except that they operate on double precision numbers.

DP18 supports many of the numerical functions that Applesoft does: SIN, COS, TAN, LOG, EXP, SGN, ABS, INT, SQR, ATN, VAL, and the string function STR\$. There is also a special function, PI, which has no arguments. You don't even write parentheses after it. You just use it like it was a constant. Wherever you use it, you get the value pi accurate to 20 digits.

#### Explanation of the Code

As in previous installments of this series on DP18, I cannot show everything at once. A whole series of subroutines which have either already been printed or will be printed in future installments are represented in this listing by ".EQ \$FFFF" in lines 1330-1550. All the data areas actually used in the code listed this month are included, so that you can see what the code is working with and on.

As mentioned above, the "&" statement sends Applesoft to line 2430. Lines 2430-2500 check for "DP" following the ampersand. If not "DP", then lines 2370-2390 branch to the next ampersand interpreter in your chain. If you have not set up another &-interpreter, then the SYNTAX ERROR message will pop out.

DP.NEXT.CMD (lines 2520-2800) begins by looking for a colon or end-of-line. End of line means you are through with DP18, so an RTS carries you back to the Applesoft interpreter. A colon means you are ready with a DP18 statement. If the next character is also a colon, however, you are sent back to Applesoft (lines 2570-2580). Next I check for the three legal tokens (IF, INPUT, and PRINT) and branch accordingly.

Since IF is simple and IF is included in this listing, let's look at IF now. Lines 3130-3280 handle the IF statement. First I evaluate the expression, which is considered to be a logical expression with a true-or-false value. Zero means false, non-zero means true. Following the expression I must find either a THEN or GOTO token. The truth value is found in DAC.EXPONENT, because a \$00 exponent means a zero value. AS.IF.JUMP in the Applesoft ROMs can handle the rest, because the THEN or GOTO pops us out of DP18 back to normal Applesoft. Neat!

Meanwhile, back in DP.NEXT.CMD, if the statement is not IF-INPUT-PRINT it must be an assignment statement. If I am successful at getting a variable name next, it may be either a DP18 variable or a string variable. If AS.VALTYPE is negative, it is a string variable and DP.STR takes over. If not, CHECK.DP.VAR will verify that it is a real array variable. The address is saved at RESULT, the DP18 expression evaluated, and then the answer saved at RESULT. And back to the top of DP.NEXT.CMD.

DP.STR handles statements like A\$=STR\$(xxx) where xxx is a DP18 expression. You can probably follow the comments in this section.

GET.A.VAR checks to see that the current character from your program is a letter, because all variables must start with a letter. If so, AS.PTRGET will search the variable tables and return with an address in the Y- and A-registers. CHECK.DP.VAR compares this address with the beginning of the array variable table. If it is inside the array table, and if the variable is real (not string or integer), it is a valid DP18 variable.

DP.EVALUATE cracks and calculates a DP18 expression. A special stack is used for temporary values, and it is deep enough to hold 10 of them. If your expression is so complicated that more than 10 temporary values need to be stacked (very unlikely), then the FORMULA TOO COMPLEX message will scream. Applesoft uses the hardware stack in page 1 for the same purpose, but it only has to stack 5-byte values; DP18 stacks 12 bytes for each value. EVALUATE starts by emptying the stack, zeroing a parenthesis level count, and clearing the accumulator (DAC). After DP.EXP finishes all the dirty work, The stack must be empty and the parenthesis level zero or there was a SYNTAX ERROR.

Actually parsing and computing an expression can be done in many ways. I chose a recursive approach that breaks the job up into little independent pieces small enough to understand. First, let's allow all expressions to be a series of relational expressions connected with ANDs and ORs. The simplest case of this is merely a relational expression alone. And the simplest relational expression is an expression all by itself with no relations. If the expression does have relational operators or ANDs or ORs, the result will be a true or false value. If not, it will have a numerical value.

Comment blocks atop DP.EXP, DP.RELAT, DP.SUM, etc. show the continued breakdown of parts of an expression. DP.RELAT connects one or more sums with relational operators. DP.SUM connects one or more terms with "+" and "-" operators. DP.TERM connects one or more factors with "\*" and "/" operators. DP.FACTOR connects one or more elements with the exponentiation operator (^). DP.ELEMENT cracks a constant, searches for a variable's value, calls a function, or calls on DP.EXP recursively to handle an expression in parentheses. DP.ELEMENT also handles the unary operators "+", "-", and "NOT".

If DP.ELEMENT determines that the element is a function call, there are several types. The VAL function is supervised by lines 5800-5830. Since the argument of the VAL function is a string expression, it is significantly different from the other functions. The ATN function is also given special treatment, because DP18 allows the ATN function to be called with one or two arguments. All the rest of the functions have one DP18 expression for an argument, so they are handled as a group. A table of addresses at lines 2160-2310 directs us to the appropriate processor. The code for all these functions will be revealed in future installments.

DP.VARNUM is called upon to handle variables and numbers. First lines 6130-6280 check for and handle the special DP18 constant "PI". Lines 6300-6350 handle DP18 variables, and lines 6370-6470 handle numbers.

PUSH.DAC.STACK pushes the 12-byte value in DAC on the special expression stack, unless there is not enough room. POP.STACK.ARG pulls a 12-byte value off the stack and plops it into ARG.

And Next Month...

There are three major areas left for future installments: INPUT, PRINT, and the math functions. Some of you have been diligently studying and entering each installment as we go, and are gradually obtaining a powerful package. Others are waiting for the Quarterly Disks, to conserve their fingertips. Remember, all the source code each three months is available on disk for only \$15.

1000 \*SAVE S.DP18 AMPER-LINK

1010 \* OR \$B03

1020 \* APPLESORT SUBROUTINES

1030 \* ADDON ADD (Y) TO TTYPR

1040 \* ADDN .EQ \$D998 ADD (Y) TO TTYPR

1050 \* ADDP .EQ \$D9A0 HANDLE TYP FOR IF

1060 \* ADDU .EQ \$D9A4 EVAL FP FOR

1070 \* ADDV .EQ \$D9A8 FORMULA

1080 \* ADDW .EQ \$D9B0 EVAL FP FOR

1090 \* ADDX .EQ \$D9B4 CHECK FOR {

1100 \* ADDY .EQ \$D9B8 CHECK FOR }

1110 \* ADDZ .EQ \$D9C0 CHECK FOR COMMA

1120 \* ADDB .EQ \$D9C4 CHARACTER SCAN OR FAIL

1130 \* ADDC .EQ \$D9C8 SYNTAX ERROR

1140 \* ADDD .EQ \$D9D2 FIND VARIABLE

1150 \* ADDE .EQ \$D9D6 LETTER CHECK

1160 \* ADDF .EQ \$D9E0 FORMULA TOO COMPLEX" ERROR

1170 \* ADDG .EQ \$D9E4 GET SPACE FOR STRING

1180 \* ADDH .EQ \$D9E8 MOVE STRING

1190 \* ADDI .EQ \$D9F2

1191 \* ADDJ .EQ \$D9F6

1192 \* ADDK .EQ \$D9F8

1193 \* ADDL .EQ \$D9F9

1194 \* ADDM .EQ \$D9F9

1195 \* ADDN .EQ \$D9F9

1196 \* ADDO .EQ \$D9F9

1197 \* ADDP .EQ \$D9F9

1198 \* ADDQ .EQ \$D9F9

1199 \* ADDR .EQ \$D9F9

1200 \* ADDS .EQ \$D9F9

1201 \* ADDT .EQ \$D9F9

1202 \* ADDU .EQ \$D9F9

1203 \* ADDV .EQ \$D9F9

1204 \* ADDW .EQ \$D9F9

1205 \* ADDX .EQ \$D9F9

1206 \* ADDY .EQ \$D9F9

1207 \* ADDZ .EQ \$D9F9

1208 \* ADDB .EQ \$D9F9

1209 \* ADDC .EQ \$D9F9

1210 \* ADDD .EQ \$D9F9

1211 \* ADDF .EQ \$D9F9

1212 \* ADDG .EQ \$D9F9

1213 \* ADDH .EQ \$D9F9

1214 \* ADDI .EQ \$D9F9

1215 \* ADDJ .EQ \$D9F9

1216 \* ADDK .EQ \$D9F9

1217 \* ADDL .EQ \$D9F9

1218 \* ADDM .EQ \$D9F9

1219 \* ADDN .EQ \$D9F9

1220 \* ADDO .EQ \$D9F9

1221 \* ADDP .EQ \$D9F9

1222 \* ADDQ .EQ \$D9F9

1223 \* ADDR .EQ \$D9F9

1224 \* ADDS .EQ \$D9F9

1225 \* ADDT .EQ \$D9F9

1226 \* ADDU .EQ \$D9F9

1227 \* ADDV .EQ \$D9F9

1228 \* ADDW .EQ \$D9F9

1229 \* ADDX .EQ \$D9F9

1230 \* ADDY .EQ \$D9F9

1231 \* ADDZ .EQ \$D9F9

1232 \* ADDB .EQ \$D9F9

1233 \* ADDC .EQ \$D9F9

1234 \* ADDD .EQ \$D9F9

1235 \* ADDF .EQ \$D9F9

1236 \* ADDG .EQ \$D9F9

1237 \* ADDH .EQ \$D9F9

1238 \* ADDI .EQ \$D9F9

1239 \* ADDJ .EQ \$D9F9

1240 \* ADDK .EQ \$D9F9

1241 \* ADDL .EQ \$D9F9

1242 \* ADDM .EQ \$D9F9

1243 \* ADDN .EQ \$D9F9

1244 \* ADDO .EQ \$D9F9

1245 \* ADDP .EQ \$D9F9

1246 \* ADDQ .EQ \$D9F9

1247 \* ADDR .EQ \$D9F9

1248 \* ADDS .EQ \$D9F9

1249 \* ADDT .EQ \$D9F9

1250 \* ADDU .EQ \$D9F9

1251 \* ADDV .EQ \$D9F9

1252 \* ADDW .EQ \$D9F9

1253 \* ADDX .EQ \$D9F9

1254 \* ADDY .EQ \$D9F9

1255 \* ADDZ .EQ \$D9F9

1256 \* ADDB .EQ \$D9F9

1257 \* ADDC .EQ \$D9F9

1258 \* ADDD .EQ \$D9F9

1259 \* ADDF .EQ \$D9F9

1260 \* ADDG .EQ \$D9F9

1261 \* ADDH .EQ \$D9F9

1262 \* ADDI .EQ \$D9F9

1263 \* ADDJ .EQ \$D9F9

1264 \* ADDK .EQ \$D9F9

1265 \* ADDL .EQ \$D9F9

1266 \* ADDM .EQ \$D9F9

1267 \* ADDN .EQ \$D9F9

1268 \* ADDO .EQ \$D9F9

1269 \* ADDP .EQ \$D9F9

1270 \* ADDQ .EQ \$D9F9

1271 \* ADDR .EQ \$D9F9

1272 \* ADDS .EQ \$D9F9

1273 \* ADDT .EQ \$D9F9

1274 \* ADDU .EQ \$D9F9

1275 \* ADDV .EQ \$D9F9

1276 \* ADDW .EQ \$D9F9

1277 \* ADDX .EQ \$D9F9

1278 \* ADDY .EQ \$D9F9

1279 \* ADDZ .EQ \$D9F9

1280 \* ADDB .EQ \$D9F9

1281 \* ADDC .EQ \$D9F9

1282 \* ADDD .EQ \$D9F9

1283 \* ADDF .EQ \$D9F9

1284 \* ADDG .EQ \$D9F9

1285 \* ADDH .EQ \$D9F9

1286 \* ADDI .EQ \$D9F9

1287 \* ADDJ .EQ \$D9F9

1288 \* ADDK .EQ \$D9F9

1289 \* ADDL .EQ \$D9F9

1290 \* ADDM .EQ \$D9F9

1291 \* ADDN .EQ \$D9F9

1292 \* ADDO .EQ \$D9F9

1293 \* ADDP .EQ \$D9F9

1294 \* ADDQ .EQ \$D9F9

1295 \* ADDR .EQ \$D9F9

1296 \* ADDS .EQ \$D9F9

1297 \* ADDT .EQ \$D9F9

1298 \* ADDU .EQ \$D9F9

1299 \* ADDV .EQ \$D9F9

1300 \* ADDW .EQ \$D9F9

1301 \* ADDX .EQ \$D9F9

1302 \* ADDY .EQ \$D9F9

1303 \* ADDZ .EQ \$D9F9

1304 \* ADDB .EQ \$D9F9

1305 \* ADDC .EQ \$D9F9

1306 \* ADDD .EQ \$D9F9

1307 \* ADDF .EQ \$D9F9

1308 \* ADDG .EQ \$D9F9

1309 \* ADDH .EQ \$D9F9

1310 \* ADDI .EQ \$D9F9

1311 \* ADDJ .EQ \$D9F9

1312 \* ADDK .EQ \$D9F9

1313 \* ADDL .EQ \$D9F9

1314 \* ADDM .EQ \$D9F9

1315 \* ADDN .EQ \$D9F9

1316 \* ADDO .EQ \$D9F9

1317 \* ADDP .EQ \$D9F9

1318 \* ADDQ .EQ \$D9F9

1319 \* ADDR .EQ \$D9F9

1320 \* ADDS .EQ \$D9F9

1321 \* ADDT .EQ \$D9F9

1322 \* ADDU .EQ \$D9F9

1323 \* ADDV .EQ \$D9F9

1324 \* ADDW .EQ \$D9F9

1325 \* ADDX .EQ \$D9F9

1326 \* ADDY .EQ \$D9F9

1327 \* ADDZ .EQ \$D9F9

1328 \* ADDB .EQ \$D9F9

1329 \* ADDC .EQ \$D9F9

1330 \* ADDD .EQ \$D9F9

1331 \* ADDF .EQ \$D9F9

1332 \* ADDG .EQ \$D9F9

1333 \* ADDH .EQ \$D9F9

1334 \* ADDI .EQ \$D9F9

1335 \* ADDJ .EQ \$D9F9

1336 \* ADDK .EQ \$D9F9

1337 \* ADDL .EQ \$D9F9

1338 \* ADDM .EQ \$D9F9

1339 \* ADDN .EQ \$D9F9

1340 \* ADDO .EQ \$D9F9

1341 \* ADDP .EQ \$D9F9

1342 \* ADDQ .EQ \$D9F9

1343 \* ADDR .EQ \$D9F9

1344 \* ADDS .EQ \$D9F9

1345 \* ADDT .EQ \$D9F9

1346 \* ADDU .EQ \$D9F9

1347 \* ADDV .EQ \$D9F9

1348 \* ADDW .EQ \$D9F9

1349 \* ADDX .EQ \$D9F9

1350 \* ADDY .EQ \$D9F9

1351 \* ADDZ .EQ \$D9F9

1352 \* ADDB .EQ \$D9F9

1353 \* ADDC .EQ \$D9F9

1354 \* ADDD .EQ \$D9F9

1355 \* ADDF .EQ \$D9F9

1356 \* ADDG .EQ \$D9F9

1357 \* ADDH .EQ \$D9F9

1358 \* ADDI .EQ \$D9F9

1359 \* ADDJ .EQ \$D9F9

1360 \* ADDK .EQ \$D9F9

1361 \* ADDL .EQ \$D9F9

1362 \* ADDM .EQ \$D9F9

1363 \* ADDN .EQ \$D9F9

1364 \* ADDO .EQ \$D9F9

1365 \* ADDP .EQ \$D9F9

1366 \* ADDQ .EQ \$D9F9

1367 \* ADDR .EQ \$D9F9

1368 \* ADDS .EQ \$D9F9

1369 \* ADDT .EQ \$D9F9

1370 \* ADDU .EQ \$D9F9

1371 \* ADDV .EQ \$D9F9

1372 \* ADDW .EQ \$D9F9

1373 \* ADDX .EQ \$D9F9

1374 \* ADDY .EQ \$D9F9

1375 \* ADDZ .EQ \$D9F9

1376 \* ADDB .EQ \$D9F9

1377 \* ADDC .EQ \$D9F9

1378 \* ADDD .EQ \$D9F9

1379 \* ADDF .EQ \$D9F9

1380 \* ADDG .EQ \$D9F9

1381 \* ADDH .EQ \$D9F9

1382 \* ADDI .EQ \$D9F9

1383 \* ADDJ .EQ \$D9F9

1384 \* ADDK .EQ \$D9F9

1385 \* ADDL .EQ \$D9F9

1386 \* ADDM .EQ \$D9F9

1387 \* ADDN .EQ \$D9F9

1388 \* ADDO .EQ \$D9F9

1389 \* ADDP .EQ \$D9F9

1390 \* ADDQ .EQ \$D9F9

1391 \* ADDR .EQ \$D9F9

1392 \* ADDS .EQ \$D9F9

1393 \* ADDT .EQ \$D9F9

1394 \* ADDU .EQ \$D9F9

1395 \* ADDV .EQ \$D9F9

1396 \* ADDW .EQ \$D9F9

1397 \* ADDX .EQ \$D9F9

1398 \* ADDY .EQ \$D9F9

1399 \* ADDZ .EQ \$D9F9

1400 \* ADDB .EQ \$D9F9

1401 \* ADDC .EQ \$D9F9

1402 \* ADDD .EQ \$D9F9

1403 \* ADDF .EQ \$D9F9

1404 \* ADDG .EQ \$D9F9

1405 \* ADDH .EQ \$D9F9

1406 \* ADDI .EQ \$D9F9

1407 \* ADDJ .EQ \$D9F9

1408 \* ADDK .EQ \$D9F9

1409 \* ADDL .EQ \$D9F9

1410 \* ADDM .EQ \$D9F9

1411 \* ADDN .EQ \$D9F9

1412 \* ADDO .EQ \$D9F9

1413 \* ADDP .EQ \$D9F9

1414 \* ADDQ .EQ \$D9F9

1415 \* ADDR .EQ \$D9F9

1416 \* ADDS .EQ \$D9F9

1417 \* ADDT .EQ \$D9F9

1418 \* ADDU .EQ \$D9F9

1419 \* ADDV .EQ \$D9F9

1420 \* ADDW .EQ \$D9F9

1421 \* ADDX .EQ \$D9F9

1422 \* ADDY .EQ \$D9F9

1423 \* ADDZ .EQ \$D9F9

1424 \* ADDB .EQ \$D9F9

1425 \* ADDC .EQ \$D9F9

1426 \* ADDD .EQ \$D9F9

1427 \* ADDF .EQ \$D9F9

1428 \* ADDG .EQ \$D9F9

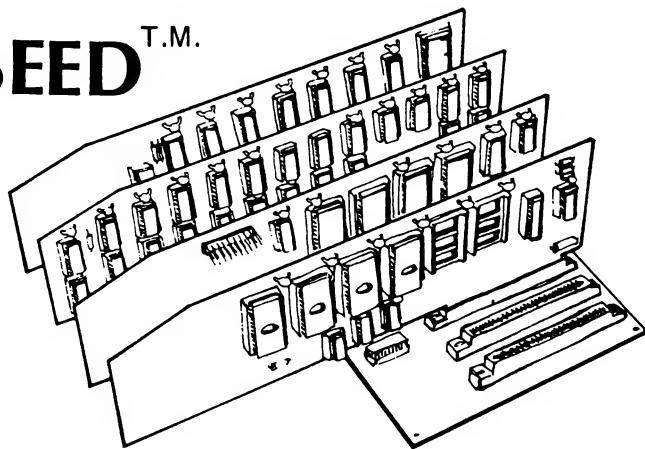
1429 \* ADDH .EQ \$D9F9

1430 \* ADDI .EQ \$D9F9

1431 \* ADDJ .EQ \$D9F9

1432 \*

# APPLESEED<sup>T.M.</sup>



Appleseed is a complete computer system. It is designed using the bus conventions established by Apple Computer for the Apple ][. Appleseed is designed as an alternative to using a full Apple ][ computer system. The Appleseed product line includes more than a dozen items including CPU, RAM, EPROM, UART, UNIVERSAL Boards as well as a number of other compatible items. This ad will highlight the Mother board.

### BX-DE-12 MOTHER BOARD

The BX-DE-12 Mother board is designed to be fully compatible with all of the Apple conventions. Ten card slots are provided. Seven of the slots are numbered in conformance with Apple standards. The additional three slots, lettered A, B and C, are used for boards which don't require a specific slot number. The CPU, RAM and EPROM boards are often placed in the slots A, B and C.

The main emphasis of the Appleseed system is illustrated by the Mother Board. The absolute minimum amount of circuitry is placed on the Mother Board; only the four ICs which are required for card slot selection are on the mother board. This approach helps in packaging (flexibility & smaller size), cost (buy only what you need) and repairability (isolate and fix problems through board substitution).

Appleseed products are made for O.E.M.s and serious industrial/scientific users. Send for literature on the full line of Appleseed products; and, watch here, each month, for additional items in the Appleseed line.

Appleseed products are not sold through computer stores.

Order direct from our plant in California.

Apple is a registered trademark of Apple Computer, Inc.

DOUGLAS ELECTRONICS

718 Marina Blvd., San Leandro, CA 94577 • (415) 483-8770





OAAD- FO 05 4910 BEQ #2<sup>TMN.SLASH</sup> / ?  
 OAA1- C9 C8 4920 CMP #2<sup>TMN.SLASH</sup> / ?  
 OAA1- FO 02 4930 BEQ #3  
 OAB2- 60 4940 CLC CC FOR \* .CS, FOR /  
 OAB2- 18 4950 .3 CLC SAVE \* OR / FLAG  
 OAB2- 08 4960 .2 PHP PUSH.DIC.STACK  
 OAB6- C1 0B 4970 JSR AS.CHRGET  
 OAB9- 20 B1 00 4980 JSR DP.FACTOR  
 OABC- 20 DD 0B 4990 JSR POP.STACK.ARGS  
 OABC- 28 5010 BCS 4 GET \* OR / FLAG  
 OAC3- B0 06 5020 JSR DIVULT .../  
 OAC5- 20 FF 5030 JSR DIV  
 OAC6- 4C AD 5040 JSR DIV  
 OACCB- 20 FF 5050 JSR DIV  
 OACE- 4C AB 5060 JSR DIV  
 5080 \* FACTORS OF A TERM  
 5090 \* FACTOR = ELEMENT \* ELEMENT  
 5110 \* FACTOR = FACTOR \* ELEMENT  
 5120 DP.FACTOR  
 OAD1- 20 B7 00 5130 JSR AS.CHRGET  
 OAD1- 20 B7 04 5130 JSR DP.ELEMENT.1  
 OAD7- 20 B7 00 5140 JSR AS.CHRGET  
 OADA- C9 CC 5150 .1 JSR DP.ELEMENT.  
 OADC- F0 01 5160 CMP #1.MPOWER ?  
 OADE- F0 01 5170 BEQ .2  
 OADE- 20 B7 00 RTS NO  
 OADF- 20 B7 00 5180 JSR PUSH.DIC.STACK  
 OAE2- 20 BE 0A 5190 .2 JSR DP.ELEMENT  
 OAE2- 20 BE 0A 5210 JSR POP.STACK.ARGS  
 OAE5- 20 DD 0B 5220 JSR DP.POWER  
 OAE5- 20 DD 0B 5230 JSR DP.POWER  
 OAE5- 4C D7 0A 5240 JSR DP.POWER  
 5250 \* ELEMENTS OF A FACTOR, VARIABLE, OR FUNCTION()  
 5260 \* ELEMENT NUMBER, VARIABLE, OR FUNCTION()  
 5270 \* ELEMENT (EX) TRUE, PUT 1 IN DAC  
 5280 \* ELEMENT NUMBER, VARIABLE, OR FUNCTION()  
 5290 \* ELEMENT = "W" OR "-" OR "NOT"  
 5310 DP.ELEMENT  
 OAEF- 20 B1 00 5320 JSR AS.CHRGET  
 OAEF- 20 B1 00 5330 DP.ELEMENT 1  
 OAF1- C9 C8 5340 CMP #1.PLUS  
 OAF3- FO F9 5350 BEQ DP.ELEMENT  
 OAF5- C9 C9 5360 CMP #1\_MINUS  
 OAF7- 20 BE 0A 5370 BNE 1  
 OAF7- 20 BE 0A 5380 LDA AD.DC.SIGN  
 OAF7- 49 FF 5390 EOR #\$FF  
 OAF7- 49 FF 5400 STA DAC.SIGN  
 OBO4- 60 5410 RTS  
 OBO4- 60 5420 JSR AS.CHRGET  
 OBO5- C9 28 5430 CHECK FOR (EXP)  
 OBO5- 60 5440 .1 CMP #1  
 OBO5- 60 5450 BNE 2  
 OBO5- 60 5460 INC RPAREN.CNT...NO  
 OBO9- C9 08 5470 INC RPAREN.CNT...NO  
 OBO9- C9 08 5480 JSR AS.CHRGET GET 1ST CHAR OF EXP  
 OBO9- C9 08 5490 JSR AS.CHRGET EXP  
 OBO12- 20 BB 5500 JSR AS.CHRCLS  
 OBO4- 60 5510 DEC RPAREN.CNT  
 OBO4- 60 5520 TAY VARIOUS FUNCTIONS  
 OBI9- A8 5530 .2 SEE IF FUNCTION  
 OBI10- 5540 C9 08 TRY NUMBER OR VARIABLE  
 OBI10- 5550 C9 08 "NOT"  
 OBI10- 5550 C9 08 CMP #1  
 OBI10- 5550 C9 08 BNE .1  
 OBI10- 5550 C9 08 INY (TTPTR), Y  
 OBI10- 5550 C9 08 CMP #1  
 OBI10- 5550 C9 08 BNE .1  
 OBI10- 5550 C9 08 INY .1  
 OBI10- 5550 C9 08 Y=2

```

OB80- B1 B8 6220 LDA (TTTPTR),Y
OB80- C9 28 6230 CMP #1,MUST NOT BE ARRAY
OB81- F0 03 6240 BEQ 1
OB81- D9 6250 JSR AS, ADDON
OB81- DF 6250 ADVANCE TXTPTR PAST "PI"
OB82- A9 FD 6260 LDA #CON.PI
OB82- 03 6270 LDY #CON.PI
OB84- AC FF 6280 JMP MOVE.YA,DAC.1 GET PI INTO DAC W/GUARD DIGITS
-----  

OB89- 20 00 6290 LDA (TTTPTR),Y DAC.1 GET PI INTO DAC W/GUARD DIGITS
OB89- 70 B0 6300 .1 JSR AS, CHRGOT
OB89- 09 6310 JSR AS, ISLETC
OB89- 09 6320 BCC 1
-----  

OB91- 20 00 6320 NOT LETTER, TRY NUMBER
OB91- D9 6330 JSR AS, P STREET
OB91- DF 6340 JSR AS, CHICK DE VAR
OB91- 03 6350 BE.SAVE IT IS REAL ARRAY
OB91- FF 6350 GET VALUE INTO DAC
-----  

OB94- 20 00 6360 JSR MOVE.YA,DAC.1 GET PI INTO DAC W/GUARD DIGITS
OB94- 70 B0 6370 .1 JSR AS, CHRGOT
OB94- 09 6380 JSR AS, ISLETC
OB94- 09 6390 BEQ 1
-----  

OBAC- C9 25 6390 CHECK FOR NUMBER
OBAC- F0 10 6400 JSR #TEN.PLUS.PLUS?
OBAC- C9 28 6410 JSR #TEN_MINUS_MINUS?
OBAC- F0 0C 6420 BEQ 1
OBAC- C9 C9 6430 JSR #TEN_MINUS_MINUS?
OBAC- F0 C9 6440 BEQ 1
OBAC- C9 30 6450 JSR #TEN_MINUS_MINUS?
OBAC- F0 12 6460 BEQ 1
OBAC- C9 34 6470 BEQ 1
OBAC- F0 FF 6480 BEQ 1
-----  

OB95- 20 00 6490 JSR MOVE.YA,DAC.1 GET PI INTO DAC W/GUARD DIGITS
OB95- 70 B0 6500 JSR MOVE.YA,DAC.1 GET PI INTO DAC W/GUARD DIGITS
-----  

OB96- 20 00 6510 PUSH.DAC,STACK
OB96- C0 12 6520 LDY STACK.PNTR
OB96- C0 12 6530 CPY #STACK_SIZE-12
OB96- A2 00 6540 BCS #2
OB96- B0 08 6550 LDX #0
OB96- B0 08 6560 STA STACK,Y
OB96- C9 30 6570 INX
OB96- C9 34 6580 INY
OB96- F0 0C 6590 BCC #1
OB96- E0 0C 6600 STY STACK.PNTR
OB96- F0 04 6610 RTS
OB96- B0 08 6620 .2 FORMULA TOO COMPLEX
OB96- B0 08 6630 .2 FORMULA TOO COMPLEX
OB96- 30 E4 6640 .2 FORMULA TOO COMPLEX
OB96- 30 E4 6650 .2 FORMULA TOO COMPLEX
-----  

OB97- 20 00 6660 POP EXPRESSION STACK INTO ARG
OB97- 70 B0 6670 JSR MOVE.YA,DAC.1 GET PI INTO DAC W/GUARD DIGITS
-----  

OB98- 20 00 6680 POP STACK,ARG
OB98- F0 08 6690 LDY STACK.PNTR
OB98- A2 08 6700 BEQ 1
OB98- A2 08 6710 .1 LDY #1
OB98- 88 6720 .1 LDY #1
-----  

OB99- 20 00 6730 LDA STACK,Y
OB99- 9D 08 6740 STA ARG,X
OB99- 9D 17 6750 DEX
OB99- 9D 17 6760 BPL #1
OB99- 9D 17 6770 STA STACK.PNTR
OB99- 9D 17 6780 RTS
-----  

OBBD- AC 08 6790 .1 HS 11100000000000000000000000000000
OBBD- F0 94 6800 .1 HS 11100000000000000000000000000000
OBBD- A2 08 6810 .1 HS 11100000000000000000000000000000
OBBD- 88 6820 .1 HS 11100000000000000000000000000000
-----  

OBEE- 20 00 6830 LDA STACK,Y
OBEE- 9D 08 6840 STA ARG,X
OBEE- 9D 17 6850 DEX
OBEE- 9D 17 6860 BPL #1
OBEE- 9D 17 6870 STA STACK.PNTR
OBEE- 9D 17 6880 RTS
-----  

OBEE- AC 08 6890 .1 HS 11100000000000000000000000000000
OBEE- F0 94 6900 .1 HS 11100000000000000000000000000000
OBEE- A2 08 6910 .1 HS 11100000000000000000000000000000
OBEE- 88 6920 .1 HS 11100000000000000000000000000000
-----  

OBEE- 20 00 6930 LDA STACK,Y
OBEE- 9D 08 6940 STA ARG,X
OBEE- 9D 17 6950 DEX
OBEE- 9D 17 6960 BPL #1
OBEE- 9D 17 6970 STA STACK.PNTR
OBEE- 9D 17 6980 RTS
-----  

OBEE- AC 08 6990 .1 HS 11100000000000000000000000000000
OBEE- F0 94 7000 .1 HS 11100000000000000000000000000000
OBEE- A2 08 7010 .1 HS 11100000000000000000000000000000
OBEE- 88 7020 .1 HS 11100000000000000000000000000000
-----  

OBEE- 20 00 7030 LDA STACK,Y
OBEE- 9D 08 7040 STA ARG,X
OBEE- 9D 17 7050 DEX
OBEE- 9D 17 7060 BPL #1
OBEE- 9D 17 7070 STA STACK.PNTR
OBEE- 9D 17 7080 RTS
-----  

OBEE- AC 08 7090 .1 HS 11100000000000000000000000000000
OBEE- F0 94 7100 .1 HS 11100000000000000000000000000000
OBEE- A2 08 7110 .1 HS 11100000000000000000000000000000
OBEE- 88 7120 .1 HS 11100000000000000000000000000000
-----  

OBEE- 20 00 7130 LDA STACK,Y
OBEE- 9D 08 7140 STA ARG,X
OBEE- 9D 17 7150 DEX
OBEE- 9D 17 7160 BPL #1
OBEE- 9D 17 7170 STA STACK.PNTR
OBEE- 9D 17 7180 RTS
-----  

OBEE- AC 08 7190 .1 HS 11100000000000000000000000000000

```

OBFD- 41 374 41  
OC003- 58 947 93  
OC006- 23 85  
6810 CON. PT  
HS 413141592653597932385

Now you can monitor and control the world (or at least your part of it) with a little help from  
**APPLIED ENGINEERING**

## APPLIED ENGINEERING

12 BIT, 16 CHANNEL,  
PROGRAMMABLE GAIN

- **PROGRAMMABLE GAIN A/D**  
In 1984, a design incorporates the latest in VLSI and IC technologies.
- Complete 12 bit A/D converter, with an accuracy of 0.02%.
- 16 single ended channels (single ended means that your signals are measured against the Apple's GND) or 8 differential channels. Most all the signals you will measure are single ended.
- 9 software programmable full scale ranges, any of the 16 channels can have any range at any time. Under program control, you can select any of the following ranges:  $\pm 10$  volts,  $\pm 5$  V,  $\pm 2.5$  V,  $\pm 1.25$  V,  $\pm 0.625$  V,  $\pm 0.3125$  V,  $\pm 0.15625$  V, or  $\pm 0.078125$  V.
- Very fast conversion (12 micro seconds).
- Analog input resistance greater than 1,000,000 ohms.
- Laser-trimmed scaling resistors.
- Low power consumption through the use of CMOS devices.
- The user connector has +12 and -12 volts on it so you can power your sensors.
- Only elementary programming is required to use the A/D.
- The entire system is on one standard size plug in card that fits neatly inside the Apple.
- System includes sample programs on disk. **PRICE \$219**

pie programs on  
**PRICE \$319**

A few applications may include the monitoring of • flow • temperature • humidity • wind speed • wind direction • light intensity • pressure • RPM • soil moisture and many more.

### 8 BIT, 8 CHANNEL A/D

- 8 Channels
- 8 Bit Resolution
- On Board Memory
- Fast Conversion (.078 ms per channel)
- A/D Process Totally Transparent to Apple (looks like memory)

The APPLIED ENGINEERING A/D BOARD is an 8 bit, 8 channel, memory buffered, data acquisition system. It consists of an 8 bit A/D converter, an 8 channel multiplexer and 8 x 8 random access memory.

The analog to digital conversion takes place on a continuous, channel sequencing basis. Data is automatically transferred to on board memory at the end of each conversion. No A/D converter could be easier to use.

Our A/D board comes standard with 0, 10V full scale inputs. These inputs can be changed by the user to 0, -10V, or -5V, +5V or other ranges as needed. The user connector has +12 and -12 volts on it so you can power your sensors.

• Accuracy: 0.3%  
• Input Resistance: 20K Ohms Typ

**PRICE \$129.00**

## SIGNAL CONDITIONER

Our 8 channel signal conditioner is designed for use with both our A/D converters. This board incorporates 8 F.E.T. op-amps, which allow almost any gain or offset. For example: an input signal that varies from 2.00 to 2.15 volts or a signal that varies from 0 to 50 mV can easily be converted to 0-10V output for the A/D.

The signal conditioner's outputs are a high quality 16 pin gold I.C. socket that matches the one on the A/D's so a simple ribbon cable connects the two. The signal conditioner can be powered by your Apple or from an external supply.

- 4.5" square for standard card cage and 4 mounting holes for standard mounting. The signal conditioner does not plug into the Apple, it can be located up to  $\frac{1}{2}$  mile away from the A/D.
- 22 pin 1.5" spacing edge card input connector (extra connectors are easily available i.e. Radio Shack).
- Large bread board area.
- Full detailed schematic included.

PRICE \$79.00

## DIGITAL INPUT/OUTPUT BOARD

- Provides 8 buffered outputs to a standard 16 pin socket for standard dip ribbon cable connection.
- Power-up reset assures that all outputs are off when your Apple is turned on.
- Features 8 inputs that can be driven from TTL logic or any 5 volt source.

The SUPER INPUT/OUTPUT board manual includes many programs for inputs and outputs. A detailed schematic is included.

**Some applications include:** Burglar alarm, direction sensing, use with relays to turn on lights, sound buzzers, start

Please see our other full page ad in this magazine for information on Applied Engineering's Timemaster Clock Card and other products for the Apple. Our boards are far superior to most of the consumer electronics made today. All I.C.'s are in high quality sockets with mil-spec. components used throughout. P.C. boards are glass-epoxy with gold contacts. Made in America to be the best in the world. All products compatible with Apple II and IIe.

Applied Engineering's products are fully tested with complete documentation and available for immediate delivery. All products are guaranteed with a no hassle three year warranty.

**Texas Residents Add 5% Sales Tax**

Send Check or Money Order to:  
**APPLIED ENGINEERING**  
**P.O. Box 798**  
**Carrollton, TX 75006**

Call (214) 492-2027  
7 a.m. to 11 p.m. 7 days a week  
MasterCard, Visa & C.O.D. Welcome  
No extra charge for credit cards

## Faster Amper-routine to Zero Arrays.....Johan Zwiekhorst Maasmechelen, Belgium

Although I have never subscribed to Apple Assembly Line, a friend of mine (who lives in nearby Heerlen, the Netherlands) does, and I always read his copies.

A few days ago I needed a routine to clear to zero all the elements in a number of Applesoft arrays, so I started looking in my friend's collection of AAL for such a program. I found the article entitled "Save Garbage by Emptying Arrays" in the December 1982 issue, pages 22-25.

That routine, however, only cleared string arrays. Bob designed it to set all strings in an array to null strings, so that garbage collection would be faster. But I needed a fast way to clear integer and real arrays as well. Bob's routine was also limited to clearing one array per call.

My routine clears any type of arrays, and can accept a list of array names separated by commas. It uses the ampersand hook, like this:

```
& CLEAR array1,array2,array3,...
```

You can load the routine in any available memory, anywhere you have a spare 79 bytes. The listing shows it assembled into the ever-popular \$300 space, but there are no internal addresses which require it to be there. Just be sure you hook the ampersand to the program, wherever you put it. If it is at \$300, hook it like this:

```
POKE 1013,76 : POKE 1014,0 : POKE 1015,3
```

The program is very similar to Bob's 1982 version: I eliminated the check he made for string arrays, added ampersand control, and checked for a comma to allow a list of array names rather than just one.

Lines 1250-1260 check that the byte following the ampersand is the CLEAR token. If not, a SYNTAX ERROR will result. If it is CLEAR, all is well.

Lines 1280-1290 check for a comma, and are not used until we have finished clearing an array. At the end, lines 1690-1710, you find my test after clearing an array. If the next byte of program is not a colon or end of line, it will branch back to the comma-test.

The code in between zeroes all the data bytes in an array. I could have done it the same way Bob did, but I did change a few things. Compare mine with his and you will learn two ways to control a clearing loop.

How about a complete example of using &CLEAR? Lets make three arrays, with a mixture of types and dimensions. Of course, when the DIM statement works it initially zeroes the arrays, but I needed them cleared again later on.

```

100 DIM A(10,20), B$(200,4,4), C$(20)
110 PRINT CHR$(4)"BLOAD B.CLEAR ARRAYS,A$300"
120 POKE 1013,76:POKE1014,0:POKE1015,3
...
500 &CLEAR A,B$,C$

```

```

1000 *SAVE S.CLEAR ARRAYS
1010 -----
1020 * &CLEAR <ARRAY LIST>
1030 * SETS ALL VALUES IN REAL ARRAYS TO 0
1040 * INTEGER ARRAYS TO 0
1050 * STRING ARRAYS TO ""
1060
1070 * WRITTEN BY JOHAN ZWIJKHORST, BASED ON
1080 * "CLEAR STRING ARRAY" BY BOB SANDER-CEDERLOF
1090 * IN DECEMBER, 1982 APPLE ASSEMBLY LINE
1100
BD- 1110 T.CLEAR .EQ $BD "CLEAR" TOKEN
1120 -----
94- 1130 ARYPT .EQ $94
9B- 1140 LOWTR .EQ $9B
9D- 1150 ARYEND .EQ $9D (= FAC)
B1- 1170 CHRGET .EQ $B1
B7- 1180 CHRGOT .EQ $B7
DEC9- 1190 SYNERR .EQ $DEC9
F7D9- 1200 GETARYPT .EQ $F7D9
1210 *
1220 .OR $300 (COULD BE ANYWHERE YOU LIKE)
1240 CLEAR.ARRAYS
0300- C9 BD 1250 CMP #T.CLEAR &CLEAR?
0302- F0 07 1260 BEQ .3 ...YES
0304- 4C C9 DE 1270 .1 JMP SYNERR
0307- C9 2C 1280 .2 CMP #$2C COMMA?
0309- DD F9 1290 BNE .1
1300 *---GET STARTING ADDRESS---
030B- 20 B1 00 1310 .3 JSR CHRGET GET NEXT CHAR (SHOULD BE LETTER)
030E- 20 D9 F7 1320 JSR GETARYPT FIND NAME/ADDRESS OF ARRAY
0311- A0 04 1330 LDY #4 COMPUTE SIZE OF PREAMBLE
0313- B1 9B 1340 LDA (LOWTR),Y # DIMENSIONS
0315- 0A 0A 1350 ASL #2, AND CLEAR CARRY
0316- 69 05 1360 ADC #5 +5 (2 FOR NAME)
0318- 65 9B 1370 ADC LOWTR (2 FOR OFFSET)
031A- 48 1380 PHA (1 FOR # DIMS)
031B- A5 9C 1390 LDA LOWTR+1
031D- 69 00 1400 ADC #0 ADD CARRY
031F- 85 95 1410 STA ARYPT+1
1420 *---GET ENDING ADDRESS---
0321- 18 1430 CLC ADD OFFSET TO GET ADDRESS OF END
0322- A0 02 1440 LDY #2
0324- B1 9B 1450 LDA (LOWTR),Y
0326- 65 9B 1460 ADC LOWTR
0328- 85 9D 1470 STA ARYEND
032A- C8 1480 INY
032B- B1 9B 1490 LDA (LOWTR),Y
032D- 65 9C 1500 ADC LOWTR+1
032F- 85 9E 1510 STA ARYEND+1
1520 *---SET UP POINTER TO START---
0331- 68 1530 PLA
0332- A8 1540 TAY
0333- A9 00 1550 LDA #0
0335- 85 94 1560 STA ARYPT
0337- A6 95 1570 LDX ARYPT+1
1580 *---LOOP TO SET ELEMENTS ZERO---
0339- 91 94 1590 .4 STA (ARYPT),Y
033B- C8 1600 INY
033C- D0 03 1610 BNE .5 ...USUALLY
033E- E8 1620 INX ...NEXT PAGE
033F- 86 95 1630 STX ARYPT+1
0341- C4 9D 1640 .5 CPY ARYEND AT END YET?
0343- D0 F4 1650 BNE .4 ...NO
0345- E4 9E 1660 CPX ARYEND+1
0347- D0 F0 1670 BNE .4 ...NO
1680 *---CHECK IF ANOTHER ARRAY---
0349- 20 B7 00 1690 JSR CHRGOT
034C- D0 B9 1700 BNE .2 ...YES, UNLESS SYNTAX ERROR
034E- 60 1710 RTS

```

Turn an Index into a Mask.....Bob Sander-Cederlof

How do you write a program that will turn a number from 0 to 7 into a bit mask \$01, \$02, ...\$40, \$80? I want an index of 0 to return \$01, 1 to return \$02, 2 to return \$04, and so on up to 7 returning \$80.

The simplest, shortest, and speediest is to use a direct table look-up. Assuming the byte with the index value is in the A-register, the code would look like this:

```
AND #7           isolate index bits
TAX             index to X-register
LDA TABLE,X    get mask from table
```

and the table would look like this:

```
TABLE .HS 01020408
.HS 10204080
```

This technique has the wonderful advantage that if you need a different translation, you can simply use a different table. For example, if you want the reverse pattern, with 0 returning \$80 and 7 returning \$01, simply change the table to:

```
TABLE .HS 80402010
.HS 08040201
```

The table lookup method has the shortest code, but counting the table does take 14 bytes. If you don't worry so much about speed and flexibility, you can write a little loop that will create the mask value like this:

```
MAKE.MASK.2
    AND #7           isolate index bits
    TAX             index into X-register
    LDA #$01         initial mask value
.1     ASL             shift loop to position
    DEX             to Xth bit
    BPL .1          shifts once to many
    ROR             restore after extra shift
    RTS
```

I put an RTS at the end because this piece of code makes a nice size subroutine. Nevertheless, for comparison to the table lookup code above, let's count neither the JSR to call it nor the RTS at the end. The shift-loop method takes only 10 bytes, four less than the table lookup. But it is slower, taking 14 cycles if the index is 0, 21 if 1, up to 63 for an index of 7. Sometimes saving four bytes is more important than speed, and sometimes speed is more important.

To generate the reverse sequence with the shift loop method, make three simple changes to MAKE.MASK.2: the initial mask value from \$01 to \$80; the ASL to LSR; and the ROR to ROL.

Note that both techniques shown above use the X-register. If the X-register is busy, you could use the Y-register instead.

Just for the challenge, I wanted to see if I could write a reasonably efficient index-to-mask routine that did not use the X- or Y- registers at all.

The first method that came to mind was fast enough, but took too much space and did not seem creative. It involved a series of CMP and BEQ instructions to branch to 8 different LDA's:

```
SILLY.WAY
    AND #7  isolate index
    BEQ .0  index=0
    CMP #1
    BEQ .1  index=1
    ...
    CMP #6
    BEQ .6  index=6
    LDA #$80  index=7
    RTS
.0    LDA #$01
    RTS
.1    LDA #$02
    RTS
...
.6    LDA #$40  index=6
    RTS
```

If I had written every line above, you would see that it takes 52 bytes.

Next I thought of a more efficient way to do the CMP's so that not so many were needed.

```
NOT.SO.SILLY.WAY
    AND #7  isolate mask
    BEQ .0  index=0
    CMP #4
    BEQ .4  index=4
    BCS .60 index=5, 6, or 7
    CMP #2  index=1, 2, or 3
    BEQ .2  index=2
    BCS .3  index=3
    LDA #$02  index=1
    RTS
.60   CMP #6  index=5, 6, or 7
    BEQ .6  index=6
    BCS .7  index=7
    LDA #$20  index=5
    RTS
.0    LDA #$01  index=0
    RTS
.2    LDA #$04  index=2
    RTS
    and so on.
```

This method takes a total of 46 bytes.

Here is one which is even shorter, which uses "tricky" arithmetic.

```
TRICKY.WAY
    AND #7
    CMP #2
    BCC .5  (0 or 1) plus 1
    BEQ .5  (2) plus CARRY plus 1 --> 4
    CMP #4
    BCC .4  (3) plus 4+1 --> 8
    BEQ .3  (4) plus 6+4+1+C --> $10
    CMP #6
    BCC .2  (5) plus $10+6+4+1
    BEQ .1  (6) plus $1E+$10+6+4+1+C
    ADC #$3F  (7) plus $3F+$1E+$10+6+4+1+C
.1    ADC #$1E
.2    ADC #$10
.3    ADC #6
.4    ADC #4
.5    ADC #1
    RTS
```

Not counting the RTS, that is 31 bytes. Cases 0 and 1 take only 9 cycles. The longest one, when the index is 7, takes 32 cycles.

All of these longer methods can be made to generate the reverse sequence by simply inverting the index before beginning the tests. Use "EOR #7" before the "AND #7".

Help Wanted

Electronic Engineer

Applied Engineering, manufacturer of  
Apple peripherals, needs a digital  
design engineer with Apple experience.

(214) 492-2027

I came up with an even trickier version, which shaved another byte or two off TRICKY.WAY. Believe it or not, it really works:

```
TRICKIER.WAY.REVERSE
EOR #7
TRICKIER.WAY
  AND #7      isolate index
  SEC        00-01-02-03-04-05-06-07
  ROL        01-03-05-07-09-0B-0D-0F
  CMP #3
  BCC .0      turn 0 into $01
  CMP #7
  BCC .12     03-->02, 05-->04
  ADC #6     ..-.-.-0E-10-12-14-16
  CMP #$12
  BCC .34     0E-->08, 10-->10
  ADC #$2B     ..-.-.-.-.-3E-40-42
  CMP #$42
  BCC .56     3E-->20, 40-->40
  ASL        42-->84-->80
  .56      AND #$E0
  .34      AND #$F8
  .12      AND #$FE
  .0       RTS
```

If the index is 0, this one takes 11 cycles. Worst case is for index 7, at 34 cycles.

A source file on the quarterly disk will include all of the above examples, plus a driving program that runs through all 8 cases and displays the results for each and every method.

In real life, I would probably use the shift-loop or the table look up. Most likely the table lookup, because it is the easiest to understand and modify, and by far the shortest in time. Nevertheless, it is very useful to experiment with other techniques. You learn a lot from the experience, and it is fun!

#### Don Lancaster's AWIle TOOLKIT

Solve all of your Applewriter™ IIc hassles with these eight diskette sides crammed full of most-needed goodies including . . .

- Patches for NULL, shortline, IIc detrashing, full expansion
- Invisible and automatic microjustify and proportional space
- Complete, thorough, and fully commented disassembly script
- Detailed source code capturing instructions for custom mods
- Clear and useful answers to hundreds of most-asked questions
- Camera ready print quality secrets (like this ad, ferinstance)
- New and mind-blowing WPL routines you simply won't believe
- Self-Prompting (!) glossaries for Diablo, Epson, many others
- Includes a free "must have" bonus book and helpline service

All this and bunches more for only \$39.95. Everything is unlocked and unprotected. Order from SYNERGETICS, 746 First Street, Box 809-AAL, Thatcher, AZ, 85552. (602) 428-4073. VISA or MC accepted.

Put Your Messages on the Screen.....William M. Reed

COUT is slow. COUT with DOS looking on is even slower. And I suppose with ProDOS, more so. If you want to get a short message on the screen in a hurry, you can bypass COUT and put it there directly.

In all of the following examples I am going to assume that the message is stored in RAM exactly as it should be on screen, and that after the last character is a byte with \$00 in it. I also assume that you are only writing one line, so that the message will not spill over to another line.

Here is a loop that writes a message on the bottom line of the screen:

#### MESSAGE

```
LDY #0
.1  LDA MESSAGE,Y
    BEQ .2      ...END OF MESSAGE
    STA $7D0,Y
    INY
    BNE .1      ...ALWAYS
.2  RTS
```

If you want to write on the current line, whose base address is kept by the monitor in BASL and BASH (\$28 and \$29), just change the STA \$7D0,Y line to STA (BASL),Y.

All well and good for 40 columns, but what about the 80-column //e and //c screens? Well, you can still do it, like this:

#### MESSAGE.80

```
LDX #0      MESSAGE INDEX
.1  TXA
    LSR      COLUMN/2, ODD/EVEN TO CARRY
    TAY      INDEX INTO SCREEN MEMORY
    LDA MESSAGE,X
    BEQ .3      ...END OF MESSAGE
    STA PAGE1
    BCS .2      ...ODD, PAGE 1
    STA PAGE2      ...EVEN, PAGE 2
.2  STA (BASL),Y
    INX
    BNE .1      ...ALWAYS
.3  RTS
PAGE1 .EQ $C054
PAGE2 .EQ $C055
```

Of course, these routines put the messages on the screen only. But that may be just what you want, to put messages on the screen without affecting the report going out to file or printer. Also, these routines do not handle CR, end of line, scroll, etc; but they sure get to the screen in a hurry!

## Bibliography on Apple Hi-Res Graphics.....Bob Sander-Cederlof

There has been a lot of material published in the last seven years about Apple's hi-res graphics. The problem is finding it! Most of the neat programs and explanations have not yet made it from the pages of various magazines into full size books. I recently decided to make a list, so that I don't have to keep leafing through mile-high stacks of magazines. Since I have never been a devotee of Pascal, I purposely omitted most articles relating to graphics in that language. I also omitted reviews and announcements of commercial hi-res products.

I looked through my book shelves and noted all books I could find there. I also went through all my back issues of Byte, Micro, Call APPLE, and Apple Orchard. Still to go are Nibble, Kilobaud, Softalk, and Creative Computing.

### Books

**Apple Graphics & Arcade Game Design**, Jeffrey Stanton. The Book Co., 1982, 288 pages, \$19.95. By the time you work through this one, you have a functioning hi-res arcade game!

**Apple II Graphics**, Ken Williams. Prentice Hall, 1983, 150 pages, \$19.95. (Originally a series of articles in Softline Magazine, Sep 81 through Jan 83.)

**Applied Apple Graphics**, Pip Forer. Prentice-Hall, 1984, about 400 pages plus diskette, price unknown. Lo-res, Hi-res, 3-D, etc., with over 50 program in BASIC on disk.

**Graphically Speaking**, Mark Pelczarski. Softalk Books, 1984, 170 pages, \$19.95. Originally a series of articles which ran from May 1982 through September 1983 in Softalk Magazine. Includes many programs in Applesoft and assembly language. Covers drawing, animation, filling, packing/unpacking, and 3-D. Disk available.

**Microcomputer Graphics**, Roy E. Myers. Addison-Wesley, 1982, 282 pages, \$12.95. More than 80 Applesoft programs. 2-D and 3-D graphics, windowing, transformations, hidden lines, and much more. Disk available.

### Books with some material on Apple Graphics

**Animation, Games, and Sound for the Apple II/III**, Tony Fabbri. Prentice-Hall, 144 pages.

**Enhancing Your Apple II, Volume I**, Don Lancaster. Howard Sams & Co., 1984, 268 pages, \$15.95. Hardware and software tricks for switching between modes and screens dynamically, programs for hundreds of hi-res colors and patterns, fast screen fill. Good explanations of the way things work.

**What's Where in the Apple**, William F. Luebbert. Micro Ink, 1982, about 300 pages. First half of book is text describing Apple; chapter 14 covers lo-res graphics, and chapter 16 covers hi-res graphics. Includes details about hardware switches, memory mapping, and firmware.

### Magazine columns

**Assembly Lines**, Roger Wagner, Softalk Magazine. From March 82 to June 83 this column covered various topics in Apple hi-res graphics. It should be made into a book, but has not yet been.

**The Graphics Page**, Bill Budge, Softalk Magazine. Oct 83 through Jun 84. Deep material, by the author of Pinball Construction Set. Further installments were promised, but not yet seen.

**Apple II Graphics**, Ken Williams, Softline Magazine. Sep 81 through Jan 83. Now available in book form (see above).

**Graphically Speaking**, Mark Pelczarski, Softalk Magazine. May 82 through Sep 83. Now available in book form.

### Magazine Articles

#### Byte

**Apple FAX: Weather Maps on a Video Screen**, Keith H. Sueker. Jun 84, 146-151.

**CHEdit: a Graphics-Character Editor**, Jerry Sweet. May 82, p426-444.

**Double the Apple II's Color Choices**, Robert H. Sturges. Nov 83, p449-463.

**Double-Width Silentype Graphics for Apple**, Charles Putney. Feb 82, p413-423.

**GRPRINT: an Apple Utility Program**, Douglas Arnott. Dec 82, 398-403.

**Interactive 3-D Graphics for Apple II**, Andrew Pickholtz. Nov 82, 474-505.

**Kinetic String Art for the Apple**, Louis Cesa. Nov 80, p62-63.

**More Colors for your Apple**, Allen Watson, Steve Wozniak. Jun 79, p60-68.

**New Shape Subroutine for the Apple**, Richard T. Simoni. Aug 83, p292-309.

**Picture Perfect Apple**, Phil Roybal. Jan 81, p226-235.

**Shape Table Conversion for the Apple II**, Dave Partyka. Nov 79, p63.

**Simplified Theory of Video Graphics**, Allen Watson. Part 1, Nov 80, p180-189. Part 2, Dec 80, p142-156.

**Three-dimensional Graphics for the Apple II**, Dan Sokol, Nov 80, p148-154.

#### Micro

**Apple Bits**, Richard C. Vile Jr. Part I, Sep 81, p75-77. Part 2, Oct 81, p94-96. Part 3, Nov 81, p105-108. (Lo-Res)

**Apple Color Filter**, Stephen R. Berggren. Jun 81, p53-54.

**A Hi-Res Graph Plotting Subroutine in Integer BASIC for the Apple II**, Richard Fam. Feb 80, p9-10.

**Apple Graphics, staff.** Sep 81, p49. Intro to several other articles.

**Apple Graphics for Okidata Microline 80, Gary Little.** May 83, p80-86.

**Apple Hi-Res Graphics and Memory Use, Dan Weston.** Nov 82, p79-81.

**Apple II High Resolution Graphics Memory Organization, Andrew H. Eliason.** Oct-Nov 1978, p43-44.

**Apple II Hi-Res Picture Compression, Bob Bishop.** Nov 79, p17-24.

**Apple Pascal Hi-Res Screen Dump, Robert D. Walker.** Feb 83, p54-55.

**Apple Shootdown, a lo-res graphics game, Eric Grammer.** Nov 82, p72-73.

**A Versatile Hi-Res Graphics Routine for the Apple, Adam P. King.** Mar 83, p77-81.

**Constructing Truly 3-D Mazes, Dr. Alan Stankiewicz.** Aug 84, p19-21.

**Creating Shape Tables, Improved!, Peter A. Cook.** Sep 80, p7-12.

**Define Hi-Res Characters for the Apple II, Robert F. Zant.** Aug 79, p44-45.

**Getting Around the Apple Hi-Res Graphics Page, Eagle Berns.** Nov 82, p93-95.

**Graphing Rational Functions, Ron Carlson.** Dec 80, 7-9.

**Hi-Res Characters for Logo, Dan Weston.** Sep 83, p50-53.

**Hi-Res Screen Dump for Epson MX-80, Robert D. Walker.** Apr/May 84, p55-61.

**How to Do a Shape Table Easily and Correctly, John Figueiras.** Dec 79, p11-22.

**Introduction to 3-D Rotation on the Apple, Chris Williams.** Nov 82, p99-101.

**Paddle Hi-Res Graphics, Kim G. Woodward.** Sep 81, p68-69.

**Random Number Generator in Machine Language for the Apple, Arthur Matheny.** Includes a graphics simulation of a globular cluster. Aug 82, p57-60.

**SHAPER: A Utility Program for Managing Shape Tables, Clement D. Osborne.** Sep 81, p50-56.

**Sun and Moon on the Apple, Svend Ostrup.** Jan 83, p35-37. Hi-res simulation of orbits and phases.

**True 3-D Images on Apple II, Art Radcliffe.** Sep 81, p71-73.

**Call-A.P.P.L.E.**

**80-column //e Lo-Res Graphics, Rob Moore.** Jul 83, p9-13.

**Adding XPLLOT to Applesoft, Mark Harris.** Apr 84, p17-18,24.

**A Higher Text Apple-cation, Donal Buchanan.** Nov 82, p47-50. Using Higher Text for ancient alphabets.

**Animation with Data Arrays, Pat Connolly. Nov-Dec 80, p11-17.**

**Apple Gaming: Playing Card Generation, Jim Hilger. Nov-Dec 79, p39-45; Jan 80, p39. Hi-res playing card pictures from Integer BASIC.**

**Applesoft Firmware Card Hi-Res Routines, Steve Alex. Oct 79, p33.**

**Applesoft Graphics Mover, Homer O. Porter. Sep 83, p29-31.**

**Arcade Graphics Techniques, Chris Jochumson. Apr 83, p9-14.**

**Character Generator ROM, Ian M. Jackson. Nov 82, p21-29. Programs for moving a Higher Text font into ROM.**

**Color 21, Darrell Aldrich. Jul-Aug 79, p21.**

**Color Me Apple, M. A. Iannce. Nov 82, p9-18. In-depth explanation of hi-res color with demo program.**

**Doing the Splits, Roy Myers. Aug 82, p61-65. Making room for hi-res pictures by moving your program.**

**Graphic Garbage Collection, Richard Cornelius & Melvin Zandler. Nov 82, p53-55. Lets you watch garbage collection activity on the hi-res screen.**

**Higher Text in Action, Steve Brugger. Jan 84, p30-31.**

**Higher Text on the Loose, Val J. Golding. Jun 81, p47-49. Explanation of the background functions of Higher Text.**

**Hi-Res Dump program modification, Tom Lewellen. Jul-Aug 79, p36.**

**Hi-Res Full Scroll, Edward C. So. Feb 82, p23-34. Scroll up, down, left, or right by one pixel position at a time.**

**Hi-Res Hi-Jinx, Edward C. So. Apr 82, p59. POKEing and PEEKing dots.**

**Hi-Res Screen Switch (program), Wes Huntress. Jul-Aug 80, p48-49.**

**Hi-Res Slide Show, Stowe Keller. Dec 83, p49-54.**

**Magic Square Dance, J. Taylor. Sep 83, p51-52.**

**MX-100 Hi-Res Dump, Bruce C. Detterich. Mar 82, p41-49.**

**Painting the RAMcard, Donald W. Miller Jr. Apr 83, p51-54.**

**Picture Compression, Edward C. So. May 82, p21-35. Very complete, builds on Bob Bishop's attempts.**

**Playing Card Generator, Vincent Aderente. Nov 82, p31-35. Applesoft versions of Jim Hilger's stuff.**

**Sorunch, Darrell Aldrich. Jun 79, p21-23. Squeeze four pictures into one screen.**

**Shape Display Utility, Major Peter M. Beck. Mar-Apr 80, p39.**

**Shape Table Splicer**, Cyrus W. Roton. Sep 83, p33-35.

**Slow Plot**, Jim Morisset. Nov 82, p63-64. Speed control for hi-res drawing.

**Smooth Animation**, Jonathan Kandell. Feb 83, p61-62.

**The Graphics Toolkit**, Randi J. Rost. Part 1: Apr 84, p10-15 (screen mapping). Part 2: May 84, p23-26. Part 3: Aug 84, p43-48. Line drawing algorithms, disassembly of Applesoft HPLOT.

**Three Dee Demos**, David Sun. Jan 83, p49-51.

**Understanding Hi-Res Graphics**, Loy Spurlock. Jan 80, p6-15.

**Using the Splitter**, Norman L. Kushnick. Jan 83, p53-55. More help in making room for pictures.

**Why Don't You Watch Where You're Going?**, Kenneth Manly. Oct 80, p25-28. A hi-res SCRn function.

**Zoom**, Neil Konzen. Jan 80, p28-32. Expand a 1/9th screen to full size.

#### **Apple Orchard**

**Double-Size Graphics for the Silentype**, Bruce F. Field. Spring 81, p30-34.

**Hi-Res Dump Routine for Integral Data Printer (IDS-225)**, Darrell & Ron Aldrich. Mar-Apr 80, p54-55. (Originally published in Call APPLE).

**Hi-Res Graphics: Resolving the Resolution Myth**, Bob Bishop. Fall 80, p7-10.

**How the Dot Patterns Produce Colors**, Allen Watson III. Jan 84, p44-46.

**How the Double Hi-Res Hardware Came to Be**, Allen Watson III. Jan 84, p42,43.

**Notes on Hi-Res Graphics Routines in Applesoft**, C. K. Mesztenyi. Spring 81, p17-19.

**Practical Super Hi-Res Graphics**, R. H. Good. Spring 81, p20.

**Secrets of Professional Graphics**, William Harvey. Part I, Behind the Scenes, Sep-Oct 82, p64-72. Part II, The Real Challenge: Putting It All Together, Nov-Dec 82, p36-53. Part III, Techniques of Animation, Mar 83, p62-69.

**Shape Definition Conversion Table**, David G. Huffman. Fall 81, p78-79.

**Shaping Up with the Apple II**, Mark L. Crosby. Mar-Apr 80, p37-45.

**The Mysterious Orange Vertical Line**, Pete Rowe. Fall 80, p11.

**True 16-color Hi-Res**, Allen Watson III. Jan 84, p26-41.

**Understanding Hi-Res Graphics, and how to include text in your Hi-Res Graphics Programs**, Loy Spurlock. Fall 80, p12-21.

## Some Great New Books.....Bob Sander-Cederlof

"Beneath Apple ProDOS", by Don Worth and Pieter Lechner. Quality Software, 1984, 276 pages plus 10 page reference card, \$19.95. (Buy it from us for \$18 plus shipping.)

We have been waiting a long time for this one, by the authors of "Beneath Apple DOS". If you read that one, you'll want this one too. And if you use or plan to use ProDOS, you will almost REQUIRE this new book.

Apple has documented ProDOS pretty thoroughly, but just TRY to get a copy of their books. Hardly any Apple dealers stock the reference manuals now. Apple requires a minimum order to buy the manuals, and they are a relatively slow moving item. Hence, dealers don't order them. Some we have talked with lately refused to admit they knew of the existence of even the Apple //e Reference Manual (over 18 months old now)! And Apple so far will not sell the books to anyone who is not an authorized Apple dealer. Catch-22, right?

But even if you have Apple's ProDOS reference manuals, as I do, you still need "Beneath Apple ProDOS". Look at the table of contents, and see if you can resist.

The most heavily thumbed pages in my copy of "Beneath Apple DOS" are the ones which give detailed comments on the entire DOS assembly language image. Unfortunately, the equivalent section does not come bound in to "Beneath Apple ProDOS". Since Apple has decided to freeze DOS, a published commentary is possible. But ProDOS is deliberately kept warm and fluid. So far there are at least four versions around; all have the same characteristics and machine language interface, but subroutines have been shuffled and rewritten. A line-by-line commentary could become obsolete every six months.

A special coupon is bound into the book at the place where you would expect the commentary. If you want the commentary, you remove the coupon page, fill in your name, address, and ProDOS version number, and send it with \$12.50 to Quality Software. With the commentary you will receive a new coupon so you can order a subsequent supplement when ProDOS changes versions.

"Assembly Cookbook for the Apple II/IIe", Don Lancaster. Howard Sams & Co., 1984, 408 pages, \$21.95. (Buy it from us for \$20 plus shipping.)

Don is sold on the synergistic combination of a full-screen 80-column word processor for handling source code with an assembler. His favorite pairing is Applewriter //e with EDASM (from DOS ToolKit). Consequently a large section of the book is devoted to how the marriage is performed, what the advantages are, and how to work around or ignore the disadvantages. Don knows Applewriter inside out, and uses it for all his word processing as well as for programming. There are some distinct advantages to using the same editor for both: writing books about assembly language programming is easier; only one set of



## **FONT DOWNLOADER & EDITOR (\$39.00)**

Turn your printer into a custom typesetter. Downloaded characters remain active while printer is powered. Use with any Word Processor program capable of sending ESC and control codes to printer. Switch back and forth easily between standard and custom fonts. All special printer functions (like expanded, compressed etc.) apply to custom fonts. Full HIRES screen editor lets you create your own characters and special graphics symbols. Compatible with many parallel printer I/F cards. User driver option provided. For Apple II, II+, //e. Specify printer: Apple Dot Matrix, C.Ith 8510A (Prowriter), Epson FX 80/100, or OkiData 92/93.

**NEW ! ! ! The Font Downloader & Editor for the Apple Imagewriter Printer.** For use with Apple II, II+, //e (with SuperSerial card) and the new Apple //c (with builtin serial interface).

**NEW ! ! ! FONT LIBRARY DISKETTE #1 (\$19.00)** contains lots of user-contributed fonts for all printers supported by the Font Downloader & Editor. Specify printer with order.

## **DISASM 2.2e - AN INTELLIGENT DISASSEMBLER (\$30.00)**

Investigate the inner workings of machine language programs. DISASM converts machine code into meaningful, symbolic source. Creates a standard text file compatible with S-C, LISA, ToolKit and other assemblers. Handles data tables, displaced object code & even lets you substitute your own meaningful labels. (100 commonly used Monitor and Pg Zero names included.) An address-based triple cross reference table is provided to screen or printer. DISASM is an invaluable machine language learning aid to both novice & expert alike. Don Lancaster says DISASM is "absolutely essential" in his new ASSEMBLY COOKBOOK. For entire Apple II family including the new Apple //c (with all the new opcodes).

SOURCE CODE available for an additional \$30.00

## **S-C Assembler (Ver 4.0 only) SUPPORT UTILITY PACKAGE (\$30.00)**

- SC.XREF - Generates a GLOBAL LABEL Cross Reference Table for complete documentation of source listings.
- SC.GSR - Global Search & Replace eliminates tedious manual renaming of labels. Search all/part of source.
- SC.TAB - Tabulates source files into neat, readable form. SOURCE CODE available for an additional \$30.00

## **The 'PERFORMER' CARD (\$39.00)**

Plugs into any slot to convert a 'dumb' centronics-type printer I/F card into a 'smart' one. Command menu eliminates need to remember complicated ESC codes. Features include perforation skip, auto page numbering with date & title. Includes large HIRES graphics & text screen dumps. Specify printer: MX-80 with Graftrax-80, MX-100, MX-80/100 with Graftraxplus, NEC 8092A, C.Ith 8510 (Prowriter), OkiData 82A/83A with Okigraph & OkiData 92/93.

SOURCE CODE: \$30.00

## **FIRMWARE FOR APPLE-CAT: The 'MIRROR' ROM (\$25.00)**

Communications ROM plugs directly into Novation's Apple-Cat Modem card. Basic modes: Dumb Terminal, Remote Console & Programmable Modem. Features include: selectable pulse or tone dialing, true dialtone detection, audible ring detect, ring-back, printer buffer, 80 col card & shift key mod support. Uses superset of Apple's Comm card and Micromodem II commands.

SOURCE CODE: \$50.00

## **RAM/ROM DEVELOPMENT BOARD (\$30.00)**

Plugs into any Apple slot. Holds one user-supplied 2Kx8 memory chip (6116 type RAM for program development or 2716 EPROM to keep your favorite routines on-line). Maps into \$Ch00-CnFF and \$C800-CFFF.

## **NEW ! ! ! C-PRINT For The APPLE //c (\$99.00)**

Connect standard parallel printers to an Apple //c. C-PRINT is a hardware accessory that plugs into the standard Apple //c printer serial port. The other end plugs into any printer having a standard 36 pin centronics-type parallel connector. Just plug in and print! High speed data transfer at 9600 Baud. No need to reconfigure serial port or load software drivers for text printing.

Avoid a \$3.00 postage/handling charge by enclosing full payment with order. (Mastercard & VISA excluded)

**RAK-WARE 41 Ralph Road W. Orange N J 07052 (201) 325-1885**



commands, tricks, and quirks need be learned. Applewriter //e's WPL language helps overcome the disadvantages of using a screen-oriented processor on line-oriented information.

The second half of the book contains sample assembly language programs, explained in detail. These are not your run-of-the-mill examples, but great subroutines and programs you can actually use, as well as learn from.

"Microcomputer Design and Troubleshooting", by Eugene M. Zumchak. Howard Sams & Co., 1982, 350 pages, \$17.95. (Buy it from us for \$17 plus shipping.)

From time to time I am called upon to understand and work with electronics. My degree is in Electronic Engineering, but I got it in the vacuum tube era (over 20 years ago). What now fits on one chip used to fill a whole ship.... Anyway, I struggle through. But I have found a book recently that has really helped: it is not really a new book, but is new to me.

Gene Zumchak has a unique approach, which is PRACTICAL. He believes in designs which are easy to troubleshoot. He tells how adding a few low cost components here and there will avoid the expense of a logic analyzer and three weeks of debugging time. For example, using an EPROM emulator and a few LED's in critical places in a microprocessor design could save endless hours of burning and erasing EPROMs, attaching logic analyzer leads and watching oscilloscope traces, and pulling all your hair out. Although every chapter has helpful ideas in the areas of trouble prevention and diagnosis, chapter 6 is devoted entirely to the subject. Another feature Gene promotes is low power consumption.

Jack Lewis is president of Micromation, a company which makes hardware for use with the Hero-1 Robot. They have designed interfaces between Apple and Hero, speech input processors, and much more. When Jack began, he contracted with Gene Zumchak to teach his people the techniques which are now in this book. Jack is the one who recommended the book to me.

And now I recommend the book to you, if you like to dabble in hardware design. Even practicing designers will find the ideas well worth the price of reading the book.

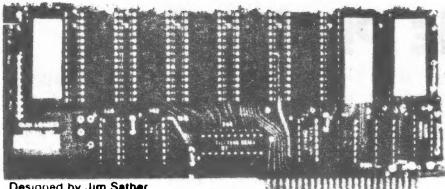
I also recommend "The Computer Journal", a monthly newsletter/magazine published by Art Carlson. \$24/year (U.S.) gets you regular articles such as "Build a 68008 CPU Board for the S-100 Bus", "Electronic Dial Indicator", "Writing Your Own Threaded Language", and "Interfacing Tips and Troubles". Write to Art at P. O. Box 1697, Kalispell, MT 59903.

# Beneath Apple ProDOS

## CONTENTS

<b>INTRODUCTION</b>	
Chapter 1	<b>TO BUILD A BETTER DOS</b>
Chapter 2	THE DEFICIENCIES OF DOS 2.1 INTERRUPTS 2.3
Chapter 3	FORMAT X FILES 2.5 WHAT YOU CAN'T WRITE 2.7 CHANGES IN DOS 2.9
<b>DISK II HARDWARE AND DISKETTE FORMATTING</b>	
Chapter 4	FORMAT AND CLOCK 3.2 TRACK FORMATTING 3.5 DISK BLOCK AND SECTOR INTERLEAVING 3.15
<b>VOLUMES, DIRECTORIES, AND FILES</b>	
Chapter 5	THE DISKETTE VOLUME 4.1 THE VOLUME DIRECTORY 4.6 FILE STRUCTURES 4.13 FILE DATA TYPES 4.19 DIR FILES—PRODOS SUBDIRECTORIES 4.26 EMERGENCY REPAIRS 4.30 FRAGMENTATION 4.33
<b>THE STRUCTURE OF ProDOS</b>	
Chapter 6	ProDOS MEMORY USE 5.1 GLOBAL PAGES 5.5 WHAT HAPPENS DURING BOOTING 5.8
<b>USING ProDOS FROM ASSEMBLY LANGUAGE</b>	
Chapter 7	CAVEAT 6.1 DIRECT USE OF THE DISKETTE DRIVE 6.2 CALLING THE DISK II DEVICE DRIVER (BLOCK ACCESS) 6.6 CALLING THE MACHINE LANGUAGE INTERFACE 6.12 MULTI-PARAMETER LISTS BY FUNCTION CODE 6.15 PASSING COMMAND LINES TO THE BASIC INTERPRETER 6.61 COMMON ALGORITHMS 6.63
<b>CUSTOMIZING ProDOS</b>	
Chapter 8	SYSTEM PROGRAMMING WITH ProDOS 7.1 INSTALLING A PROGRAM BETWEEN THE B1 AND ITS BUFFERS 7.4 ADDING YOUR OWN COMMANDS TO THE ProDOS BASIC INTERPRETER 7.5
<b>EXAMPLE PROGRAMS</b>	
Appendix A	STORING THE PROGRAMS ON DISKETTE A.3 DUMP—Track Dump Utility A.4 FORMAT—Reformat a Range of Tracks A.9 ZAP—Disk Update Utility A.19 MAP—Map FreeSpace on a Volume A.22 FIB—Find Index Block Utility A.25 TYPE—Type Command A.30 DUMBTERM—Dumb-Terminal Program
<b>DISKETTE PROTECTION SCHEMES</b>	
Appendix B	A BRIEF HISTORY OF APPLE SOFTWARE PROTECTION B.2 PROTECTION METHODS B.3 THE IDEAL PROTECTION SCHEME B.7
<b>NIBBLIZING</b>	
Appendix C	ENCODING TECHNIQUES C.1 THE ENCODING PROCESS C.5
<b>THE LOGIC STATE SEQUENCER</b>	
Appendix D	
Appendix E	ProDOS, DOS, AND SOS E.1 CONVERTING FROM DOS TO ProDOS E.1 WRITING PROGRAMS FOR ProDOS AND SOS E.3
<b>Glossary</b>	
<b>Index</b>	
<b>Reference Card</b>	

# quikLoader



Designed by Jim Sather

## SPEED

The quikLoader is the *fastest* way to load programs, **BAR NONE!** Applesoft, Integer, or machine language programs can be loaded in fractions of a second. More importantly, DOS is instantly loaded every time the computer is turned on. Integer is even loaded in the language card. This process takes less than a second, saving valuable time. The quikLoader operating system can keep track of over 250 programs stored in **PROMs** (Programmable Read Only Memory). The user simply transfers any of these programs to PROM using the instructions packed with the unit, and any PROM programmer, or we will provide this service.

## CONVENIENCE

How many times have you started to work with a frequently used program, only to find that you have misplaced the disk, or worse, had the disk damaged, or the dreaded "I/O ERROR" message flash on the screen. With the quikLoader, these nightmares can be a thing of the past. Frequently used programs are available *instantly* when you need them, without having to look for the disk, or hoping that the lengthy disk loading procedure goes smoothly. If you do need to use standard disks, the quikLoader even speeds up that process. For example, to catalog a disk, just press **ctrl-C** **Reset**. To run the "HELLO" program, press **ctrl-H** **Reset**. Other "one-key" commands include entering the monitor, booting the disk, calling up the mini-assembler, etc. The major difference between the quikLoader and the other ROM cards is the complete operating system (in PROM). This enables you to get the quikLoader catalog on the screen (by pressing **ctrl-Q** **Reset**), allowing you to see what programs are available. Loading or running of the desired program requires one keypress. Program parameters, such as starting address and length of machine language programs can be seen on the catalog screen, if desired.

## EASE OF USE

The quikLoader plugs into any slot of the **APPLE II** or **IIe**. The card is **reset** driven. To use any of the many features of the card, **RESET** is pressed in conjunction with a key. The particular key pressed chooses the feature.

## VERSATILE

The quikLoader will accept any of the popular PROMs available on the market, 2716, 2732, 2764, 27128 and 27256. These types may be freely intermixed on the card. Long programs can take up more than one PROM, or several short programs may be stored on one PROM. The quikLoader operating system even handles multiple cards, so you can easily double or triple the amount of PROM memory available. The ultimate memory capacity

of one card is 256K, so many frequently used programs and utilities can be stored. We even start your library of programs with the most popular utilities on the card, **FID** and **COPYA**. Now, if you have to copy a disk, you don't have to search for the master disk. You can start copying within 3 seconds after turning on the computer.

## INCREASED DISK CAPACITY

Since DOS is loaded from the quikLoader every time the computer is turned on, it is not necessary to take up valuable disk space with DOS. This will give you more than 5% additional space for programs and data on your disks.

## SYSTEM REQUIREMENTS

The quikLoader will work in an **APPLE II** or **IIe**. If used in a **II+**, a slightly modified 16K memory card is required in slot 0. A disk drive is required to save data.

**\$179.50**

DOS, INTEGER BASIC, FID, and COPYA are copyrighted programs of **APPLE COMPUTER, INC.** licensed to **Southern California Research Group** to distribute for use only in combination with quikLoader.

## NOW AVAILABLE FOR quikLoader:

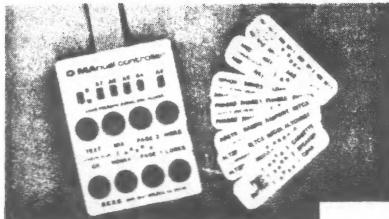
**DOUBLE-TAKE** by **BEAGLE BROS.**

**COPY ]+ by CENTRAL POINT SOFTWARE**

**BARKOVITCH I/O TRACER AND SINGLE STEP TRACE**  
**MICRO/TYPOGRAPHER BY TIDBIT SOFTWARE**

*More programs coming soon.*

## D MAnual controller



Designed by Jim Sather

This hardware product gives the user complete control over all I/O functions in the range **\$C000** through **\$C0FF**.

## EXAMPLES:

- Switch between TEXT and GRAPHICS.
- Select HIRES or LORES.
- Select Page 1 or 2.
- Turn drives ON or OFF.
- Step head either direction.
- Protect or enable language card.

All this can be done while programs are running. Commands can be issued (via push-buttons) in the middle of a program, and the desired result occurs immediately, without interfering with the normal operation of the program. The card is slot-independent, and is connected to a control panel by a four foot cable.

**\$89.95**

# SCRG PRODUCTS FOR THE APPLE COMPUTER

These items are also available from S-C Software.

quikLoader -- \$170      D MAnual Controller -- \$85.

Apple Assembly Line is published monthly by S-C SOFTWARE CORPORATION, P.O. Box 280300, Dallas, Texas 75228. Phone (214) 324-2050. Subscription rate is \$18 per year in the USA, sent Bulk Mail; add \$3 for First Class postage in USA, Canada, and Mexico; add \$12 postage for other countries. Back issues are available for \$1.50 each (other countries add \$1 per back issue for postage).

All material herein is copyrighted by S-C SOFTWARE CORPORATION, all rights reserved. (Apple is a registered trademark of Apple Computer, Inc.)